VIRTUALIZATION OF THE PDP-11/50

Joseph Curtis Winther

THESIS

VIRTUALIZATION OF THE PDP-11/50

by

Joseph Curtis Winther

and

Douglas Mark Kruse

June 1975

Thesis Advisor: B. E. Allen

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Virtualization of the PDP-11/50		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1975 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Joseph Curtis Winther and Douglas Mark Kruse		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1975 13. NUMBER OF PAGES 97
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

virtual machine
virtual machine monitor
PDP-11/50

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis presents an overview of the virtual machine concept discussing different properties of the virtual machine and the basic environment necessary to virtualize an existing computer. Concurrent with this study, feasibility of virtualizing the PDP-11/50 of the Naval Postgraduate School including problems and possible approaches were examined. These results lead to an implementation of a basic virtual machine.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
(Page 1) S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

VIRTUALIZATION OF THE PDP-11/50

by

Joseph Curtis Winther
Major, United States Marine Corps
B.S., University Of Utah, 1961

and

Douglas Mark Kruse
Captain, United States Marine Corps
B.S., Colorado State College, 1966

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1975

ABSTRACT

This thesis presents an overview of the virtual machine concept discussing different properties of the virtual machine and the basic environment necessary to virtualize an existing computer. Concurrent with this study, feasibility of virtualizing the PDP-11/50 of the Naval Postgraduate School including problems and possible approaches were examined. These results lead to an implementation of a basic virtual machine.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

The authors would like to express their gratitude to their wives for their sincere interest. Colleen and Jane deserve special recognition for their remarkable patience and support, both real and virtual, during the preparation of the thesis.

I. INTRODUCTION

During the early 1960's when the dual state third generation computers were developed, the concept of virtualization was born. Little did one recognize the potential benefits or rapid transition that would soon take place in the comouter field in the area of virtual machines.

Even though this rapid change and increased complexity may lead to the frustrations expressed by John Gosden, Vice President, Equitable Life Assurance Society [18].

> "All of these new announcements are a nuissance. We'd rather be getting on with doing productive work than trying to keep aligned with the industry's technology."

However, the virtual machine concept has arrived and is here to stay. Robert P. Goldberg [11] notes that

> "Virtual machines have finally arrived. Dismissed for a number of years as merely academic curiosities, they are now seen as cost-effective techniques for organizing computer systems resources to provide extraordinary system flexibility and support for certain unique applications."

and Phil Bookman, President, Innovation Data Processing [18] satirically comments

> "Virtual computing is complicated, but if it's simplicity you want, get a bank of 1401's."

The current state-of-the-art of virtual computing is still in its infancy and its potential is ripe for taoping, especially in an educational institution.

In the fall of 1974 and early 1975, the Naval Postgraduate School acquired two Digital Equipment Corporation PDP-11/50 computers as a part of the Signal Processing and Display Research Facility. It will also serve as a valuable tool for the computer science student.

With the enchancement of virtualization, this computer system would increase in both flexibility and potential uses. Virtual machine systems can be used to resolve a number of problems that face a student of computer science. All students could be given their own virtual computer for exploring new ideas and concepts. The student could implement and test software from the basic hardware level to the operating system level without the fear of destroying or interfering with the real system. Courses can be made available where the fundamentals of advanced software and the design and analysis of operating systems are accomplished in a 'hands-on' environment.

Besides an educational need for virtual machine's there exists the problem of security and privacy. Computer technology has spawned a whole new field of crime and generated a series of problems for both designers and users of information systems. There are truly very few systems that can be classified as secure. Particularly in the military environment, security is a ever present problem and a major concern of students who will leave this educational institute and work in a military computer installation. The virtualized computer holds great promise as the solution to this ever growing problem. It would for example, be possible

for the PDD-11/50 to integrate classified and unclassified processes without degrading the performance of the system and without the fear of possible compromise. It would also eliminate the the current locked door 'monoprocessing' environment policy for processing classified jobs.

Another need that can be solved by using a virtual computer is in system development. New enhancements to the operating system could be throughly tested and debugged in 'day light hours' and in a multiprogramming environment before being place 'on-line'. The concept of virtualization is truly an exciting area of computer science and one in which the effects on the data processing community are just beginning.

The remainder of this thesis is divided into five additional chapters.

In chapter II, the background and principles of virtualization are explored to develop a basic understanding of the virtual machine and the current terminology associated with it. A glossary (Appendix A) is provided for the convenience of the reader.

Chapter III gives a summary of the architectual design of the PDP-11/50 family of computers and highlights some of the unsuitabilities of this type of architectural design for virtualization.

Chapter IV presents the problems of virtualizing the PDP-11/50 and examines some possible solutions to these problems to include recommended milestones towards the ultimate goal of a fully virtualized PDP-11/50.

The selection of an approach to virtualize the PDP-11/50 and implementation of this approach is presented in Chapter V. A user manual (Appendix D) is also provided.

Both chapters IV and V are self-contained and those who are knowledgeable about current virtual machine applications and technology should be able to skip directly to these sections.

Chapter VI summarizes the conclusions of this implementation and contains some suggestions for further development.

II. VIRTUAL MACHINES - PRINCIPLES AND TERMINOLOGY

A. PRINCIPLES AND TERMINOLOGY

The virtual machine can basically be pictured as an imaginary copy of some real existing computer system but with the capability of executing real programs . This unique and rather new idea has introduced to the computer world the concept of multi-environmenting as compared to multiprogramming and multi-processing. In extending the basic idea of the virtual machine, one is lead to the realization that one real computer can create many imaginary computers, each of which may have different hardware characteristics and devices. In addition, each imaginary machine may support different operating systems.

"Basically a virtual machine is a very efficient simulated copy (or copies) of the bare host machine [13]". A bare host machine is simply a machine without its accompanied software, i.e. operating system. If we included the operating system's instruction set (system calls, I/O's, etc.) along with the basic hardware instructions (MOV,BR,etc.), we have introduced the instruction set for the extended machine i.e., the bare machine plus operating system is the extended machine.

The virtual machine may be further defined as an efficient, isolated, duplicate of a real machine. It first must be efficient, meaning that the programs executing under the virtual machine should basically run at speeds comparable to execution on the real machine. This attribute of a virtual

machine makes it reasonable to run production type jobs on the virtual machine.

The efficiency aspect of the virtual machine is realized by having the majority of instructions of a program running on the virtual machine executed directly on the real host computer. This concept rules out traditional emulators and complete software interpretors, for both methods demand total interpretation of all program instructions.

Secondly, the reason the virtual machine must be isolated from the rest of the system is to insure that it has complete control over managing its own systems resources, and that no unintentional interference exists between the real and virtual system.

Finally, the virtual machine must produce a duplicate or essentially identical environment of real machines to make it practical to run real jobs on them. "Since a virtual machine is a software-hardware duplicate of a real existing computer system there is always the notion of a real computer system or real machine, whose execution is functionally equivalent to the virtual machine [13]." If one must modify a job to be executed on a virtual machine because the virtual machine is slightly different from the real machine, then the virtual machine concept has loss its beauty, value, and practicality [13].

It should be obvious that certain types of instructions can not be allowed to be executed directly but must be simulated by the virtual machine. The job of identifying and trapping that type of instruction is the heart of the

virtual machine. The program that executes on the real host machine creating the virtual machine environment is called the virtual machine monitor. Its function is to be the software interface between the real system and virtual system.

Figures 1 and 2 [3] illustrate the basic differences between the conventional computer system and a virtual machine organization. The conventional dual state extended machine architecture contains only one basic machine interface which can support many user programs but is only capable of running one privileged software nucleus at a given time. As contrasted in figure 2, the virtual machine approach, the virtual machine monitor creates additional basic machine interfaces which are functionally identical to that of the real machine. Thus any privileged software nucleus that will run on the real machine will execute on the virtual machine. It is interesting to note, that the privileged software nucleus has no way of knowing whether it is being executed on the real or virtual machine.

Figure 2 should not imply that the basic machine interface supported by the virtual machine monitor must be identical to the interface of the bare machine that the virtual machine monitor runs on, i.e. a virtual IBM S/360 50 need not be produced on a real S/360 50. However, when the interface is not identical they will be, at the minimum, of the same computer family. When two interfaces are completely different (IBM S/360 versus BURROUGHS 6700) then emulation or simulation must be employed to map the one system

CONVENTIONAL EXTENDED MACHINE

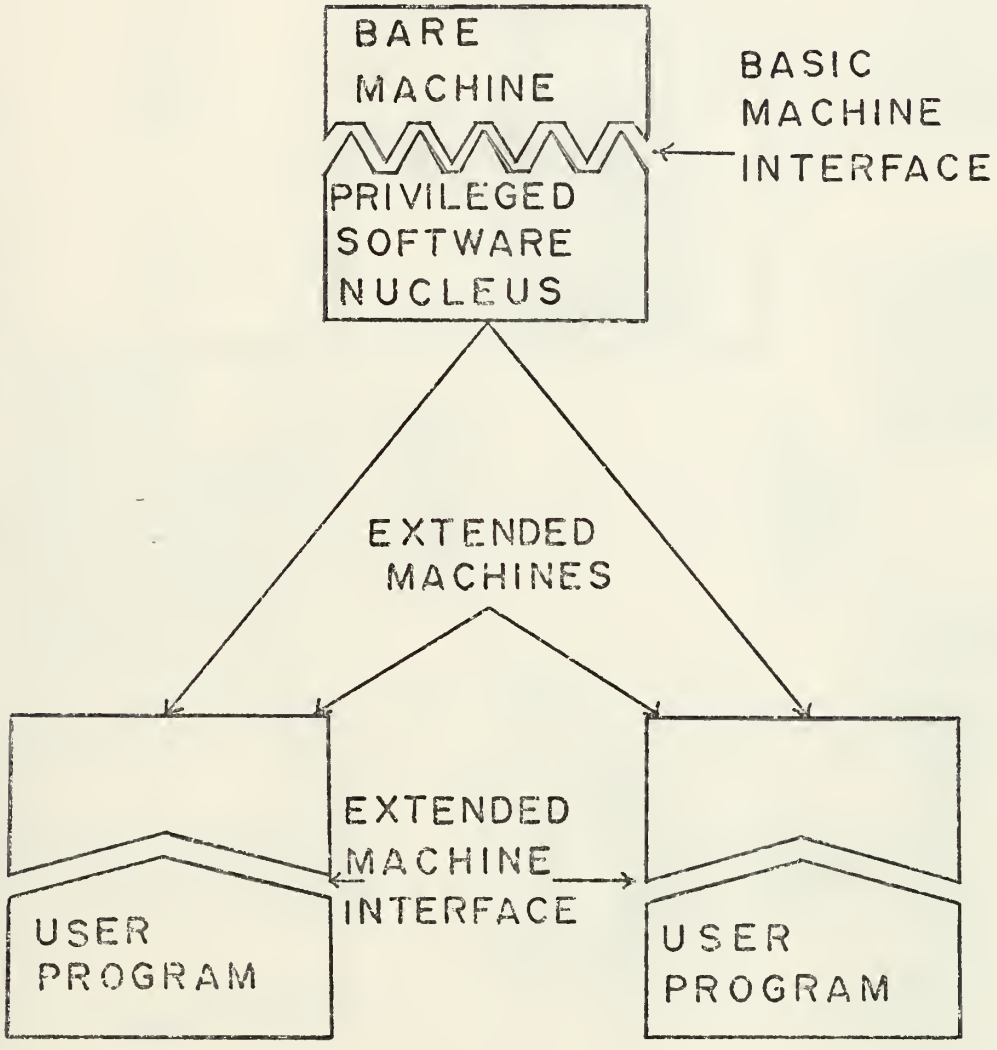


FIGURE 1

VIRTUAL MACHINE ORGANIZATION

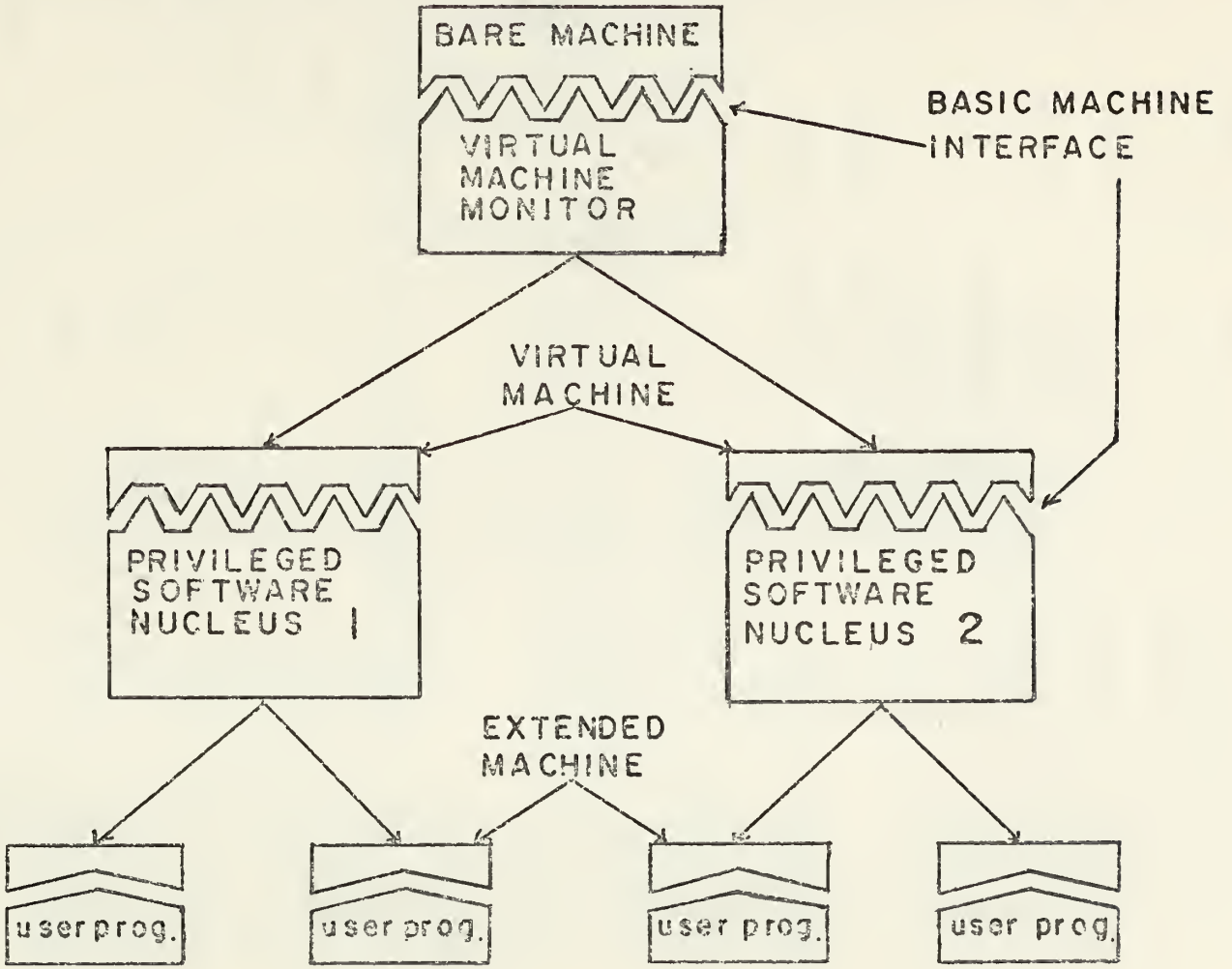


FIGURE 2

architecture into another different architecture.

The question that might be logically asked at this point is: "Are there any restrictions on the type of programs that can be processed on a virtural machine?". Surprisingly, there is basically only one type of program that will not execute properly on the virtual machine and that is a program with timing dependencies. This type of program while once popular is now extremely rare. Reference 24] provides a detailed insight into this area.

The virtual machine can now be formally defined as follows [24]:

> "A virtual computer is a hardware-software duplicate of a real existing computer system in which a statistically dominant subset of the virtual processor's instructions execute directly on the host processor in native mode."

The method of producing a virtual machine to meet the above stated definition may vary greatly. The method would depend on such things as the specific brand and model of computer, the degree of efficiency required, and the ingenuity of the programmer of the virtual machine monitor. There is no one best approach or method. The virtual machines that have been produced to date though, can be categorized into general types depending on the method of implementation [3,13,14].

A type I monitor is illustrated by figure 2. Here the virtual machine monitor runs directly on the bare machine, where as the type II monitor (figure 3) runs on the extended

TYPE II VIRTUAL MACHINE

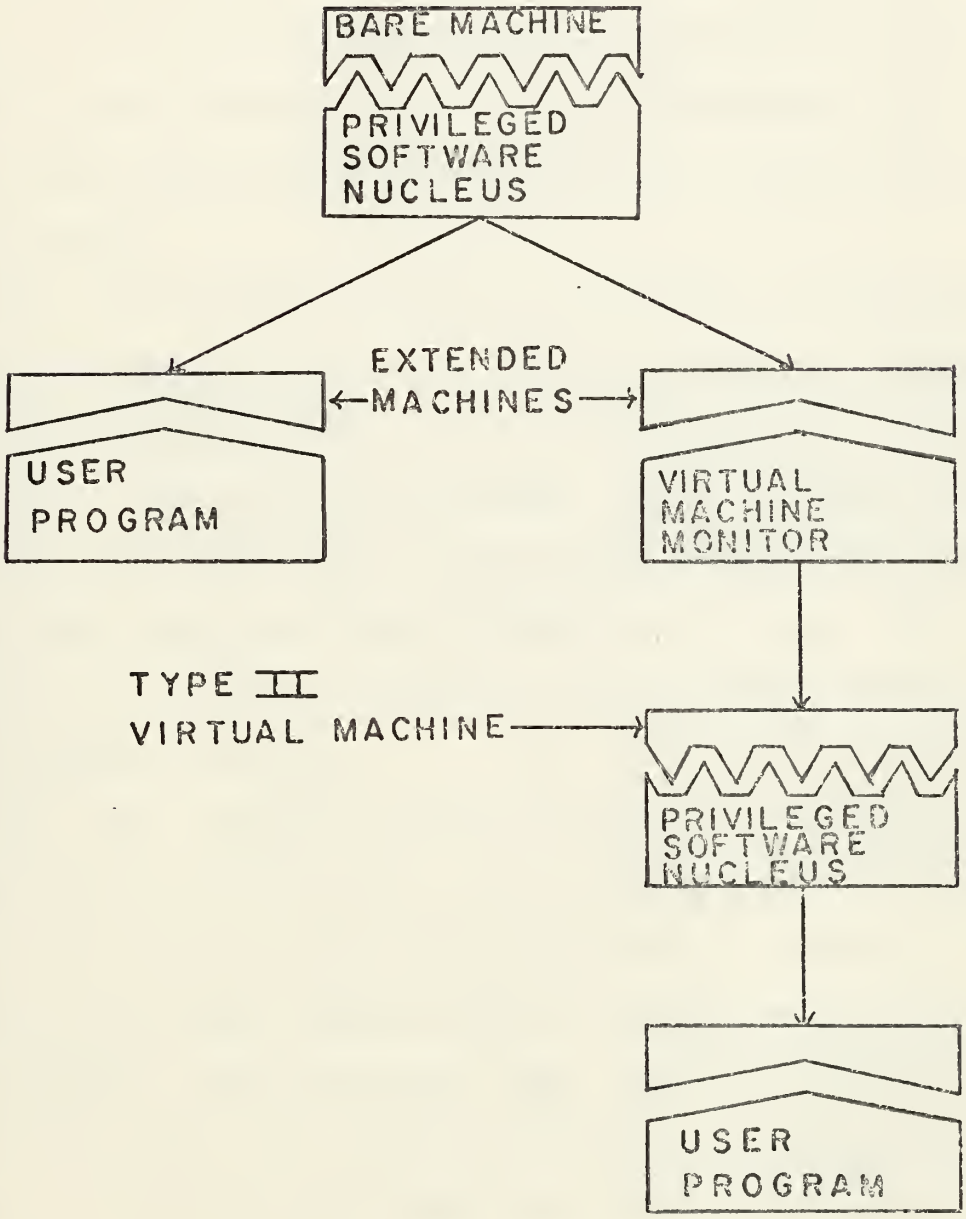


FIGURE 3

machine under an operating system.

A type I monitor's only function is to produce one or more virtual bare machines which will support any operating system that the real bare machine would support.

The type II monitor, on the other hand, can take advantage of the services provided by its operating system. For instance, such features as the memory management, the actual paging mechanism, and I/O operations of the operating system can be used.

When the virtual machine is identical to the host processor then we say that the virtual computer system is self-virtualizing. When it is not identical, then the virtual machine must, as earlier stated, be of the same processor family as the host machine and is said to be family virtualizing. An example of family virtualizing is an IBM system 370/155 as the host machine which is supporting a virtual system 360/50 machine.

One very apparent benefit of family virtualizing is in upgrading the computer in a data processing installation. Here the old system could be virtualized to process all jobs that have not been reprogrammed to meet the new systems requirements. Reprogramming then would not be a critical factor but could be accomplished in a timely, directed manner. In fact, some application programs which are processed very infrequently may never be selected for reprogramming.

Recursive virtualizing is when an additional copy or copies of the virtual machine monitor run on the virtual

system or systems that have been produced by the host computer. We have then in effect, produced another level of virtual machines. Figure 4 illustrates this recursive property, and the concepts of levels of virtual machines.

It is interesting to note that recursive virtualizing is supported only under the self-virtualizing type system. The main reason for this restriction is the fact that the virtual machine monitor is tailored to interface with a specific model of computer, and therefore, it is only possible to run the monitor when the identical interface is produced.

One can see that this property is indeed recursive, for level 2 in figure 4 is another virtual machine monitor and could have in effect recursed again and produced a third level virtual machine and so forth. Tests have successfully been conducted over a six level recursion [12].

One main advantage of practical importance in recursive virtualizing is the ability to test and modify the virtual machine monitor itself without interfering with normal processing.

B. BASIC REQUIREMENTS FOR VIRTUAL MACHINE SYSTEMS

Since the majority of instructions of application programs running on a virtual machine are going to execute directly on the host computer without any software intervention, a method must be derived to trap only those "critical instructions" that must not be allowed to be executed. Before addressing this trapping process, more fundamental questions should first be considered, namely "What is a

VIRTUAL RECURSION

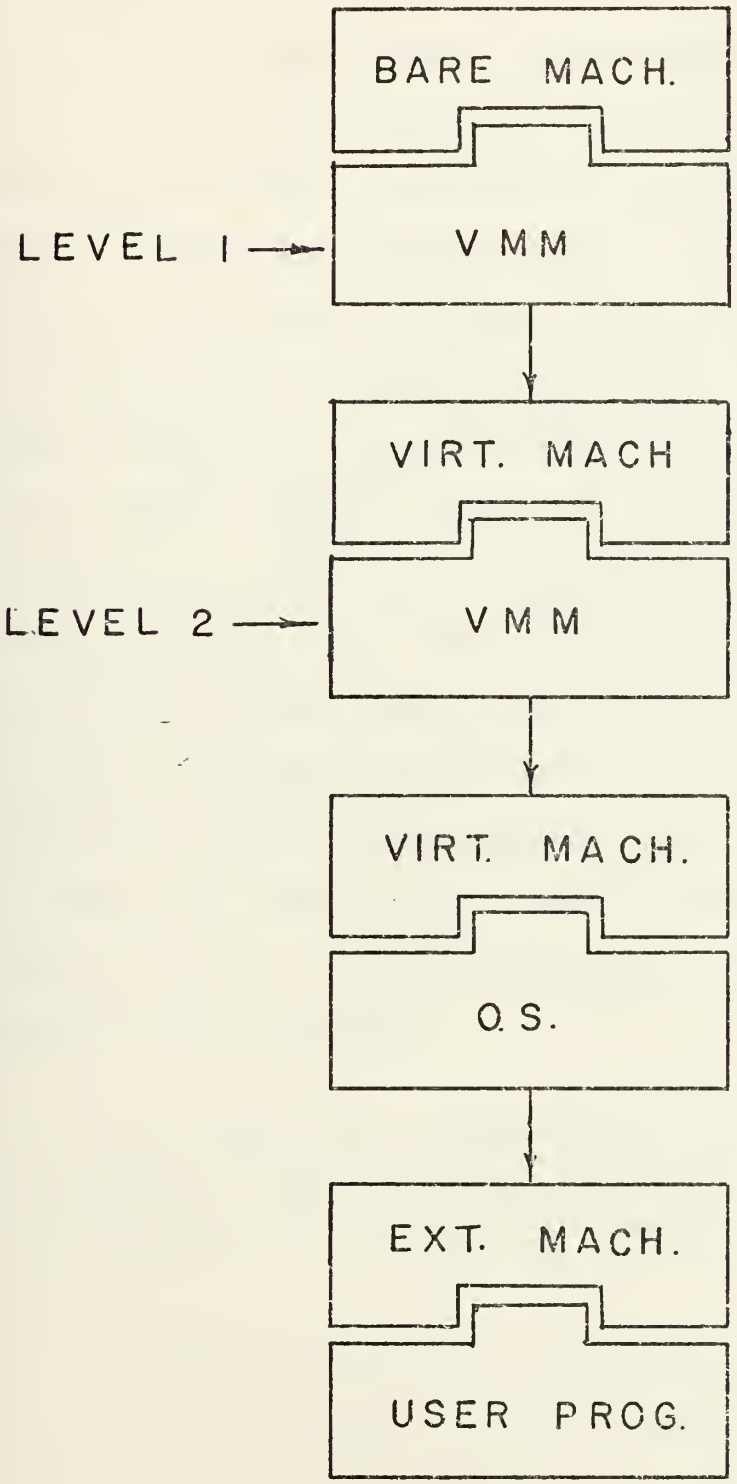


FIGURE 4

critical instruction?" and "How can one identify them?".

Basically the third generation of computers created two distinct processor modes of operation (privileged - 'supervisor' and nonprivileged - 'program'). The supervisor mode permitted certain 'privileged' instructions to be executed which are denied execution in the user mode. Privileged instructions then are those instructions which are allowed to be executed in the supervisor but not in the program mode. These privileged instructions typically control critical system features such as testing and initiating input/outout facilities or modification of address mapping mechanisms. When the user requires the execution of a privileged instruction the user program executes a supervisor call to the operating system. This privileged instruction is then executed on behalf of the user program.

Professor Robert P. Goldberg in his Ph.D thesis [13] on virtual machine architecture defined the critical instructions as the following:

> "A sensitive instruction is an instruction which, because of 3rd generation virtual machine software construction will give incorrect results if permitted to be executed directly on the host computer".

It is now obvious that sensitive instructions encountered in the virtual system cannot be allowed to execute directly, since the execution of sensitive instructions could prevent the correct interpretation of certain instructions.

He states the key to implementing a virtual machine on a

third generation system is to provide complete functional equivalence with a real machine without allowing the direct execution of sensitive instructions.

Sensitive instructions are divided into two general types, namely, control sensitive and behavior sensitive instructions.

1. Control sensitive instructions are those instructions which control the resources and environment of the system. Examples of such instructions are those which attempt to change available memory, start an I/O operation, or change the processor mode.

2. Behavior sensitive instructions are those instructions whose execution depends upon their location in real memory. An example of such an instruction is the system/360 LRA (load physical address) instruction.

The machine instruction repertoire of the host computer system must be individually analyzed to determine their sensitivity. Each instruction should be questioned as to whether its execution could produce a possible erroneous results if allowed to execute in the unprivileged mode. This identification and analysis appears to be like a fairly simple process, but in reality before this can be accomplished, a thorough understanding of both the machine language and hardware characteristics must be mastered.

Once the sensitive instructions have been identifed, it is now possible to determine if the computer system will support a virtual machine. Professor Goldberg's First Theorem on virtualizability gives us this answer [24].

> "THEOREM 1 : For any conventional third generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions."

The basic philosophy behind Theorem 1 is the following. If all sensitive instructions are a subset of the privileged instructions, then the host computer has a very convenient built-in trapping method. Any attempt to execute a privileged instruction in the program mode will generate a machine interrupt. Therefore, if the virtual machine's privileged software is running in the program state then an interrupt will be generated when the virtual machine's operating system attempts to execute any privileged instructions. Since the sensitive instruction is a subset of the privileged instructions set, it is just a matter of checking to see if the privileged instruction causing the interrupt is sensitive.

This class of interrupts can then be turned over to the virtual machine monitor for servicing. If it is a sensitive instruction that generated the interrupt then it will be simulated. If not, it will be returned to the host operating system for execution. Figure 5 illustrates this interaction. The interested reader should refer to reference [24] for proof of Theorem 1.

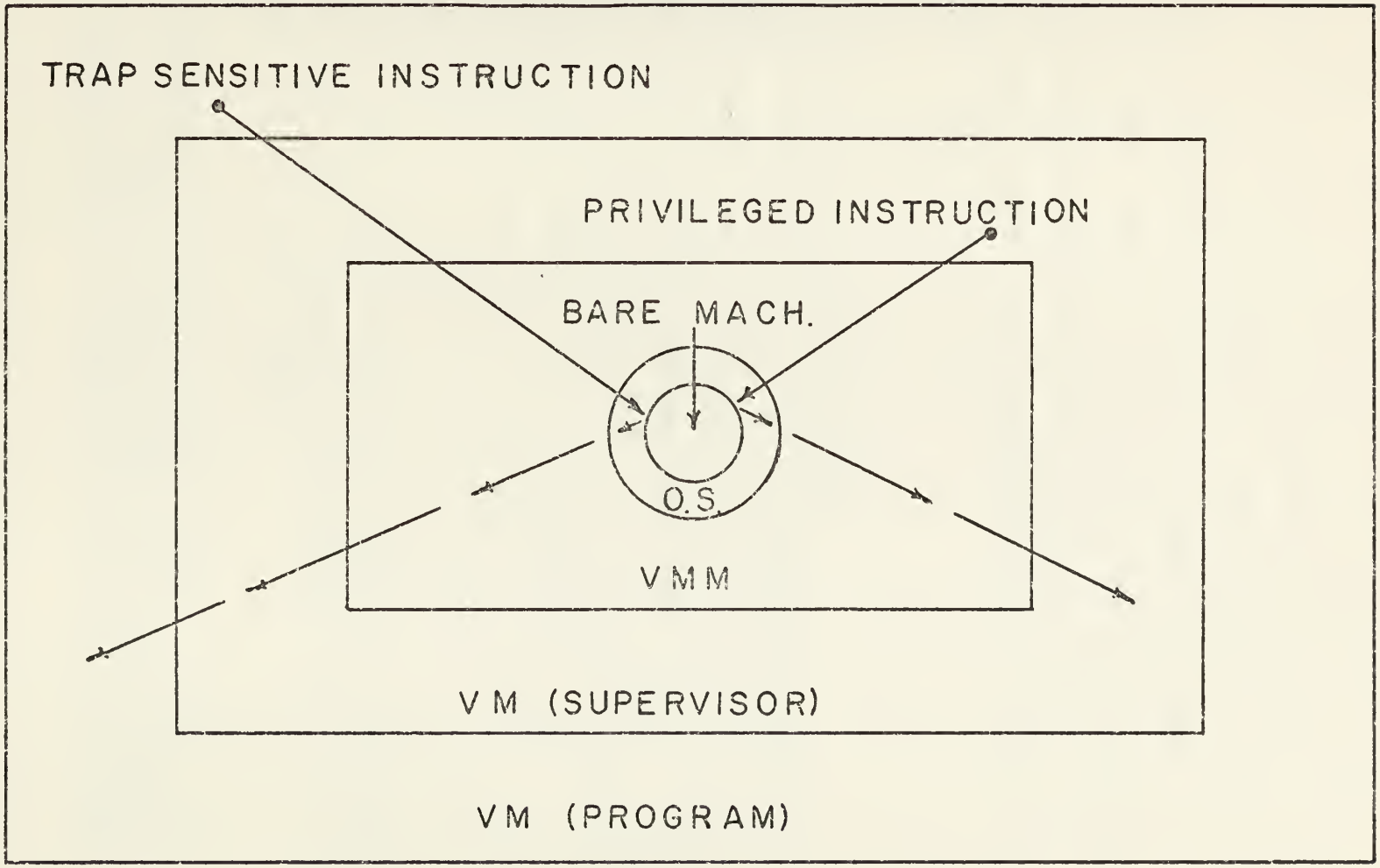

TRAP SENSITIVE INSTRUCTION
PRIVILEGED INSTRUCTION
BARE MACH.
O.S.
VMM
VM (SUPERVISOR)
VM (PROGRAM)

FIGURE 5

This rather simple theorem suprisingly proves that very few third generation architectures are virtualizable, for it only takes one instruction to cause the monitor to be forced to interrogate all instructions to prevent the execution of the one exception. This of course would cause the virtual machine to lose its efficiency and value. It should be realized, however, that third generation computers were not designed to support virtual machines, and it is their dual state nature that brought about the virtual machine concept in the first place.

C. HYBRID VIRTUAL MACHINE SYSTEMS

Since the majority of third generation computers are not virtualizable the hybrid virtual machine monitor was introduced. Here sensitive instructions are further classified into two groups depending on where they can be located, i.e. are they only located in the supervisor area of coding (supervisor sensitive) or can they be in the user area (user sensitive)?

Once the user and supervisor sensitive instructions are identified then Goldberg's Third Theorem can be applied, which is the following [24]:

> "THEOREM 3. A hybrid virtual machine monitor may be constructed for any conventional third generation machine in which the set of user sensitive instructions are a subset of the set of privileged instructions."

The main difference then between the virtual machine and the hybrid virtual machine is in the degree of instruction

interpretation. In the hybrid virtual machine all instructions, privileged or not, in the virtual supervisor mode are interpreted. User sensitive instructions will be trapped as previously mentioned.

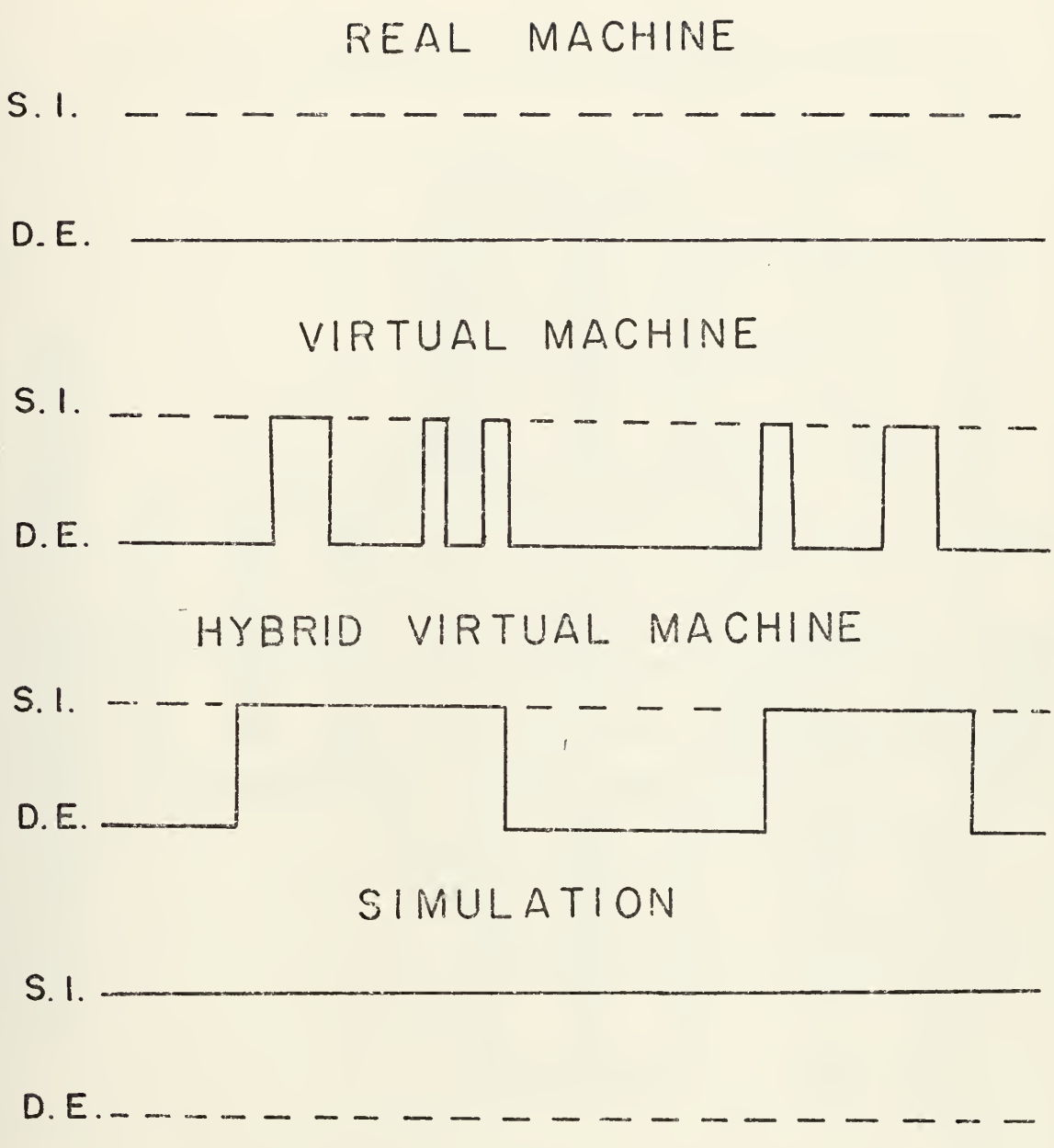
Figure 6 is an overview that compares instruction interpretation versus direct instruction execution for normal processing, virtual machine processing, hybrid processing, and for complete simulation. It should be remembered that the more time spend in interpretation the slower the execution of the job.

D. THE VIRTUAL MACHINE MONITOR

The virtual machine monitor controls the overall virtual machine system. The monitors main functions are to create the virtual machine or machines, to monitor and allocate their resources, to interrogate all trapped instructions for sensitivity, and to simulate the execution of the sensitive instructions.

The virtual machine monitor's software is composed generally of three groups of program modules plus numerous control tables (figure 7). The three groups are the dispatcher, allocator, and the interpreters. The control tables depict all the virtual machines resources available and their current status. These tables are similiar to the normal operating system's tables and control blocks.

The dispatcher is the top level control module of the virtual machine monitor. It is the initial program that receives the machine interrupts and determines if the



SOFTWARE INTERPRETION VS. DIRECT EXECUTION

FIGURE 6

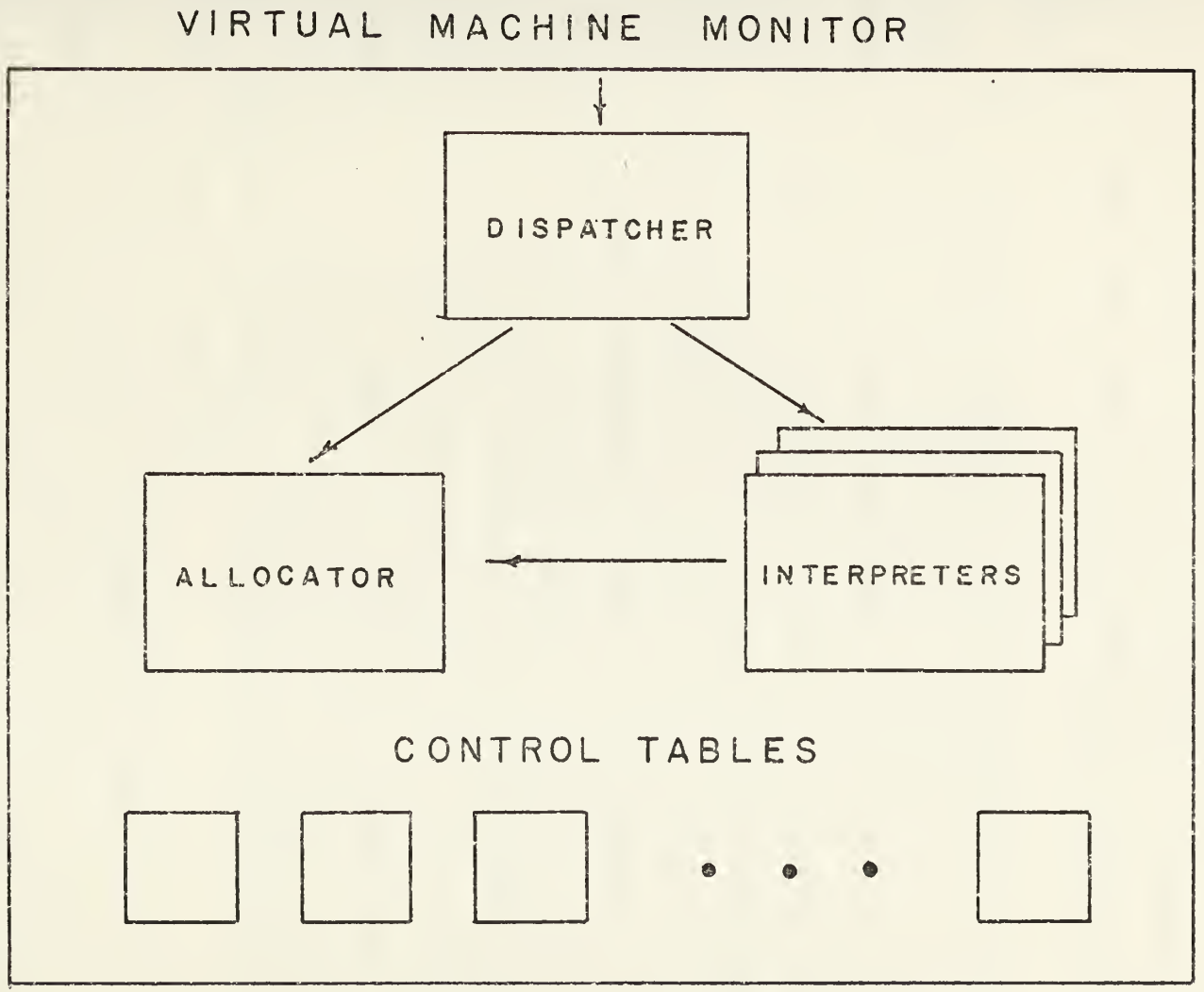

VIRTUAL MACHINE MONITOR
DISPATCHER
ALLOCATOR
INTERPRETERS
CONTROL TABLES

FIGURE 7

instruction that is trapped is sensitive. If it is sensitive then the dispatcher decides which modules to call to service this action. If it is a resource request or a resource modification, then the dispatcher will invoke the allocation module. If the action calls for a sensitive instruction to be simulated then one of the interpreter modules will be called.

The allocation module is the main keeper of the control tables for the virtual machines. It is through this module that all the virtual machine devices are created and controlled. When the virtual machine monitor is supporting multiple virtual machines, the complexity of this module is significantly increased. It now must keep track separately of the individual resources of each virtual machine.

The last group of program modules that make up the virtual machine monitor is composed of many individual modules called interpreters. There is generally one interpreter routine for each identified sensitive instruction. Each interpreter routine will simulate an instruction execution by changing the state of the virtual machine so as to reflect its actual execution.

All software - be it an operating system, application program running under the operating system, or a stand-alone program executing on the virtual machine - must execute in the nonprivileged program mode of the host computer. This is essential so that the real machine's interrupt can trap sensitive instructions. It should be obvious that a type I virtual machine monitor must execute in the supervisor

state, since it is in fact, the total operating system for the real hardware. The type II virtual machine on the other hand, has a choice between supervisor and program state.

If a type II monitor is allowed to operate in the supervisor state, then the relationship and coordination between it and the operating system must be carefully thought out and designed to insure that there is no unintentional interference. If the monitor is executed in the privileged mode, then it could execute within itself the privileged instructions that may be required to be actually executed after either being trapped or required by some interpreter module. It would thereby save the time of requesting and waiting for the operating system to execute them.

If on the other hand, the monitor is running in the program state then the interface problem with the operating system is nonexistant. All privileged instructions that are required to be executed must be turned over to the operating system. This would probably require some additional supervisor calls to be added to the system, but it would also make the virtual machine monitor smaller and less complex.

E. SOFTWARE AND HARDWARE REQUIREMENTS

The hardware and software requirements of the host computer system must meet the following criteria to adequately support a virtual computer system.

1. The method of instruction execution of non-privileged instructions in both supervisor and

program state must be roughly equivalent. This is important because the operating system of the virtual machine, which normally operates in the supervisor state, will be executing in the program state. There is also the reverse problem of the privileged instruction in the program state not causing machine interrupts. For example, in the PDP 11 family of computers certain instructions are ignored if they are encountered while in the program state.

2. A method of automatically signalling the supervisor when the virtual machine attempts to execute a sensitive instruction must be available.

3. A method of protecting the supervisor and any other virtual machine must be available to insure isolation of all systems.

4. The host computer should support a page or segmentation type virtual memory system to adequately give the virtual machines their own virtual memory required to hold its virtual operating system plus its own application programs.

F. ADVANTAGES AND APPLICATIONS OF VIRTUAL MACHINES

The advantages of a virtual machine have intentionally been left until now, because many of the benefits can be better appreciated once one has a good understanding of the concepts and workings of virtual machines. In fact, because

virtual machines lend themselves to so many practical applications, it is felt by many computer experts that the "next generation" of computers will be specifically designed to support virtual machines.

The following outlines a few of the more important advantages and applications of the virtual machine [19].

1. Software Development For The System Programs.

Because system programs cannot run under the normal operating system in a production type environment, most system programming work must be accomplished in a stand-alone environment, and generally in the middle of the night. It also causes the system to be brought down which interfers with normal processing and is wasteful of system resources, not to mention the inconvenience for both system and application programmers. Using a virtual machine for system software development eliminates that type of a working environment. System enhancement, new operating system releases, or in fact any software development can be thoroughly tested and debugged on the virtual machine before being released. All of this can be accomplished during normal working hours and without wasting system resources.

2. Elimination Of Certain Conversion Problems.

This benefit was earlier examined and can only be truly appreciated when one has lived through a mass conversion effort. The ability to be selective on what programs to convert and not to be rushed in that conversion is alone a great advantage.

3. Concurrent Running Of Dissimilar Operating

Systems By Different Users.

This application would be extremly useful to those data processing installations that are charged with the responsibility of developing application software which will be distributed to many installations and run on a variety of computer sizes and models and under different operating systems. In fact, copies of the actual operating system of the individual installation in which the program will be running could be sent to the testing installation. Then the program could be executed on a virtual machine configured to the installation and tested under their operating system.

Another benefit of running dissimilar operating systems is computer backup capability. If an installation's computer went down for an extended period of time, the installation could still process necessary jobs on a virtualized system using their backup installation's computer. They could use their own operating system and configuration thereby avoiding the difficult tasks of temporarily modifying or generating operating systems or programs generally required when one has to move their processes to another computer.

4. Testing Future Hardware.

The virtual machine gives the installation the ability to design, develop, and test programs that will be interfacing with new hardware before the actual hardware is delivered. In addition to helping in the program development, it could have been used in the hardware selection process. Different types and models of hardware devices could

have been virtualized and tested for their applicability.

5. Test Of Network Facilities.

The complicated nature of testing and implementing a computer network would be greatly simplified if many of the inter-communication software bugs can be found and corrected on the system before the actual implementation.

6. Evaluation Of Program Behavior.

The nature of the virtual machine monitor lends itself to a type of performance monitoring. Detailed interactions between the software and machine or between software modules themselves can be trapped, measured, and recorded to be later printed out for detailed analysis. Changes could be implemented and measured again to check the improvement factor before making the real modifications.

7. Security And Privacy.

The virtual computer system holds great promise in the area of security and privacy. As mentioned earlier, it is impossible for an operating system to tell if it is being executed on the real or virtual machine. Each machine virtual is completely isolated from each other so it would be impossible to spy on or try to alter any coexisting machine.

8. Education.

The ability for a student in the computer science field to be able to have his own computer and operating system is a blessing to both student and computer center manager. Here he can work with and alter on his own an operating system, without the underlying fear of destroying the real system, for the only way to truly learn about

operating systems is to actually perform system maintenance on a real system.

III. ARCHITECTURAL DESIGN OF THE PDP-11/50

This chapter is primarily designed to highlight only certain areas of the architectual design of the PDP-11 family of computers. Attention is mainly given to computer design characteristics which relates to virtualization problems. It is not intended as a complete or detailed analysis of the internal computer workings. The interested reader will find a complete and detailed description in references [7] and [8].

The PDP-11/50 is a general purpose, interrupt driven, three state computer, capable of directly addressing 28k sixteen bit words of memory without the memory management unit, and 124k words with memory management. It contains two independent sets of six general purpose registers, three stack pointer registers (one SP for each state), and a program counter register (PC).

The PDP-11/50 in addition to the optional memory management unit, supports an optional floating point processor, and a host of peripheral devices. The system is designed to support a virtual memory multiprogramming environment. The basic relationship of these different units are depicted in figure 8.

The general purpose registers are rather unique in that I/O operations do not disturb their contents and secondly, they can be used in a diversity of register operations, namely as accumulators, index registers, stack pointers, autoincrement and autodecrement registers.

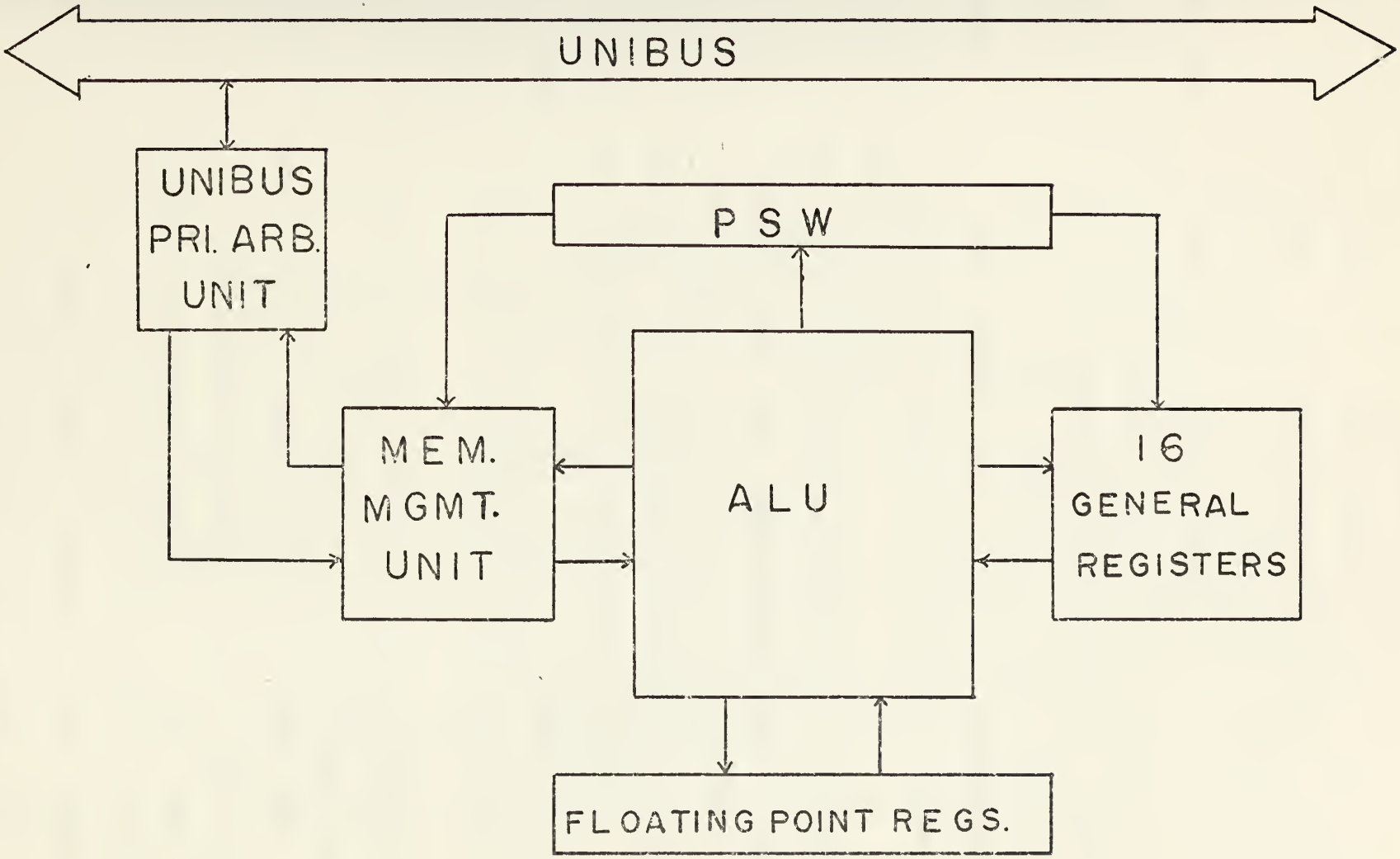

UNIBUS
UNIBUS PRI. ARB. UNIT
P S W
MEM. MGMT. UNIT
ALU
16 GENERAL REGISTERS
FLOATING POINT REGS.

FIGURE 8

The instruction set is probably the most powerful asset of the PDP-11/50 computer. The instruction set consists of over four hundred microprogrammed instructions. Instructions are so flexible that a programmer has numerous options available for coding any program routine.

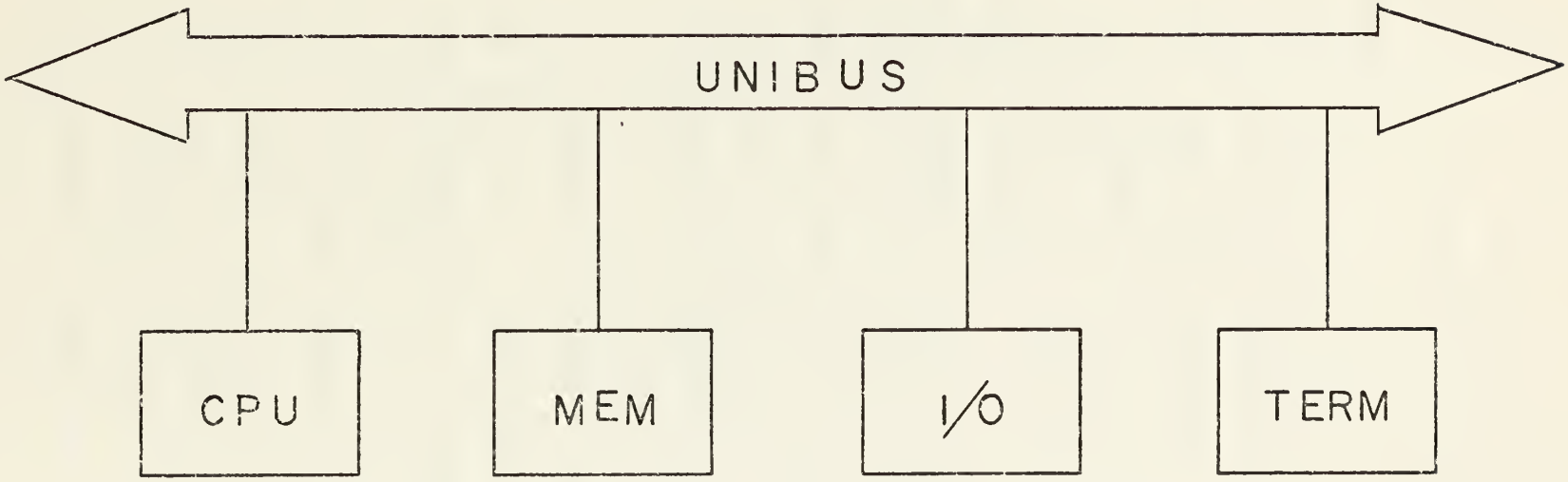
The architectural design feature of the PDD-11 family of computers that distinguishes it from other DEC systems as well as from other computer vendors is the UNIBUS. The UNIBUS is a high speed (five million bytes/seconds), asychronous, bidirectional channel that ties all system components together (figure 9).

All memory and processors are connected to the UNIBUS in a regular manner. The same addressing scheme is applied equally well to a specific device, memory or processor. This greatly simplifies I/O programming because the entire instruction set is available for input/output routines.

Machine instructions that effect the UNIBUS operation or devices attached to the UNIBUS would qualify as sensitive instructions.

The PDP-11/50 is a three state machine, as compared with the conventional two state (supervisor and program) operation. The three states or modes are kernel, supervisor, and user. Kernel mode programs are unrestricted in their use of the machine while the programs operating in the user and supervisor modes are restricted in the kind and number of instructions which may be legally executed. The supervisor mode is only slightly more powerful than the user mode.

It is interesting to note some instructions (RESET,SPL,



- STANDARD INTERFACE
- ANY ATTACMENT CAN BE UNIBUS MASTER

FIGURE 9

etc.) are ignored if not in the kernel mode while other instructions (HALT) generate an interrupt. This inconsistancy, especially when encountered in the sensitive instruction set, adds to the complexity of virtualizing the PDP-11 family of computers.

The processor status word (PS) is a special memory cell located in the upper 4k of the machine's memory. (The upper 4k is not actual memory but addresses of device registers located on the device.) The current machine operation mode at any given instance is completely determined by the current mode bits in the PS.

The PS may be changed in one of three ways: (1) Explicity, by physically modifing it via a normal machine instruction such as MOV. (2) Implicitly, through the use of one of the several machine instructions such as SPL,RTT,RTI which cause the PS to be implicity changed, and (3) Asynchronously, as a result of an interrupt or trap.

The ability to explicitly modify the PS and in fact to do any actual I/O is a function of whether or not the program's virtual address space is mapped into that portion of the machine's physical memory which contains the PS and device registers. Programs operating in the user or supervisor mode are generally restricted in this manner causing for example, I/O to be initiated only from the kernel itself. Figure 10 illustrates the real and virtual memory map.

Instructions capable of modifying the PS (SPL,RTI,RTT) also qualify as sensitive instructions. This would include a MOV instruction if allowed to access the I/O page.

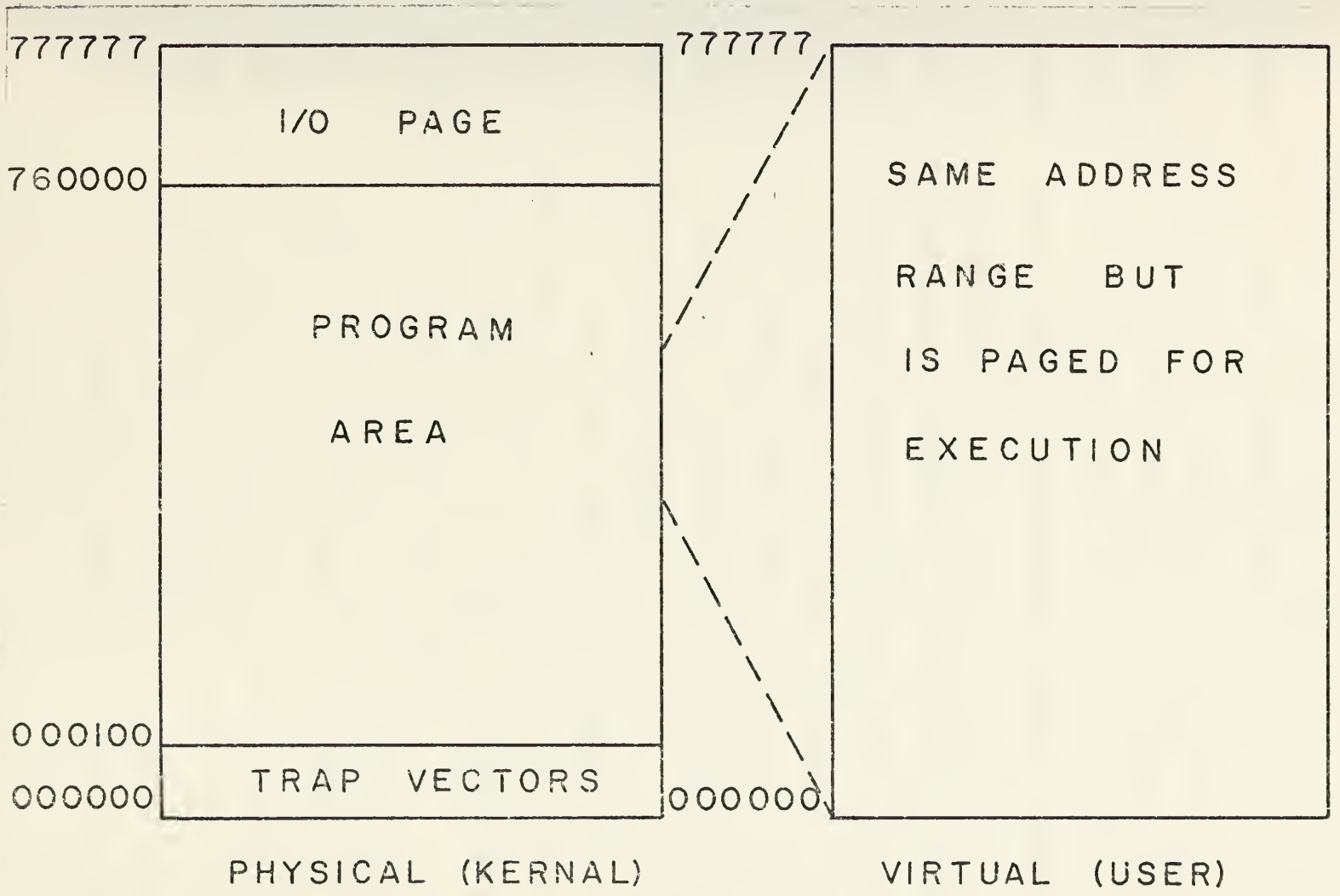

777777
I/O PAGE
760000
PROGRAM
AREA
000100
TRAP VECTORS
000000
777777
SAME ADDRESS
RANGE BUT
IS PAGED FOR
EXECUTION
000000
PHYSICAL (KERNAL)
VIRTUAL (USER)
ADDRESS SPACE

FIGURE 10

The I/O structure of the PDP-11/50 is one of the true advantages built into the system's architecture and can be one of the biggest disadvantages in virturalizing the system, depending on how the uoper 4k of virtual memory is mapped into physical address space.

It would be highly advantageous if the upper 4k virtual addresses of a program executing on the virtual machine were allowed to map directly into the physical I/O page, and the access control field of the page description register is set to abort all accesses. Thus this type of sensitive instruction is conviently trapped. If this is not the case then because of the flexibility of the instruction set and especially if the program has no restrictions on the method of accessing the I/O page, then one is lead to a one hundred percent instruction interrogation and a large increase in instruction simulation routines.

The design considerations for the asychronous interaction between modes is based around stack operations. A stack is a temporay storage area (one for each mode) used in subroutine and interrupt service linkage.

Individual program stacks can be directed to start at any virtual address and can expand either upward or downward. Any register can serve as a stack pointer. The main register used by the system for this function is the stack pointer register (SP), which by design expands downward. As ealier mentioned, there is a separate SP for each mode.

Interrupts and subroutine linkage are very similiar and only differ in their method of invocation and the registers

that are stacked and loaded. Interrupts are forced while subroutine linkage is controlled program transfer. In addition, subroutine linkage only effects the PC and not the PS. The action is basically as follows: The old PS (if caused by an interrupt) and the current PC are automatically stacked onto the new processor stack and the new linkage information (PS if required and PC) are placed in the registers.

All interrupts are required to be simulated under a virtural machine, subroutine calls are not.

This brief examination of the PDP-11/50 architecture clearly illustrates that the PDP-11/50 was not designed for supporting a vitural machine application. In fact, without significant hardware modifications a true virtural computer system can not realized.

IV. PROBLEM DEFINITION AND POSSIBLE APPROACHES

The current state-of-the-art of computer science offers a variety of approaches that may be employed to virtualize a computer system. Virtualization methods can range from extremely expensive and complicated hardware modifications to that of only minor systems changes to support a limited type II virtual machine monitor.

The first step in determining the type of virtualization approach to use is to examine the computer manager's plans and goals for the installation. Management's desires will automatically eliminate many of the possible approaches. Therefore, a brief examination of the Naval Postgraduate School's PDP-11/50 computer to include the computer's environment, primary use, and the general goals of the school towards the computer system is in order. This examination will highlight the imposed restrictions on virtualizing the system.

A. SYSTEM ENVIRONMENT

The Naval Postgraduate School has two PDP-11/50 computers and a wealth of periphial devices. These computers are not intended to serve the school in a computer service bureau fashion, but are incorported into a computer research laboratory.

The primary research areas of this computer laboratory are in the fields of operating systems design and research, signal processing, computer graphics, and hybrid computing.

Temporary configuration modifications for a specific research project offer no real problems. Peripherials can be switched between computers in matters of minutes to meet any new demands. This versitility is mainly do to the standard interface of the UNIBUS.

Managements goals though, are to have all I/O devices and computers tied together to support a real time, multiprogramming, multiprocessing operating environment. The general configuration envisioned is schematically illustrated in appendix (B). It is a general configuration, since the wide range of activities , will continually demanding configuration modifications.

A greatly enchanced version of the UNIX operating system (called MUNIX) is being designed as the primary operating system to support the desired system environment.

UNIX itself is a general purpose, multiuser, interactive operating system developed at Bell Laboratories. UNIX contains a number of sophisticated features generally only found on large systems, but with the surprising and unusual quality of simplicity and ease of use. A complete and excellent abstract of UNIX is contained in reference 28.

Even though UNIX is extremely powerful and versatile it would not meet the complete demands that would be required of the operating system. MUNIX is being developed to meet those addition demands of real-time processing.

B. VIRTUALIZATION GOALS AND PROBLEM AREAS

The Naval Postgraduate School's desired goal in the area of virtualization of the PDP-11/50 can be summarized as follows:

1. Achieve the capability to produce a virtualized PDP-11/50 computer that would support all available operating systems and be able to access any real or virtualized I/O device.

2. The virtual machine's efficiency would be sufficient to reasonably execute most types of applications.

3. Incorporate a complete debugging aid facility to assist the user in executing and programming on the virtual machine.

The decision was made that any implementation of a virtual system would be required to operate under the MUNIX system. This thereby eliminates any type I virtual machine monitor considerations.

A futher decision and restriction was that the virtual machine monitor plus programs executing on the virtual machine would execute in the user mode and in a similar manner to any other process.

This restriction imposed a serious problem on referencing the I/O page. Since the I/O page comprises the upper 4k of the address space, all programs on the virtual machine would reference this space for testing and excuting I/O. MUNIX, on the other hand, uses this virtual address space

for the user stack area. This restriction also imposes the problem of having two user mode processes executing concurrently and in harmony but at the same time having one of the processes (virtual machine monitor) be the supervisor. It was also decided that the virtualization design would if at all possible, minimize modifications to the operating system.

The main problem areas identified in both the hardware architecture and software restrictions can now be summarized.

1. That a true virtual system cannot be incorporated on the PDP-11/50 as defined by Goldberg's Theorem 1, since all identified sensitive instructions (appendix C) are not a subset of the privileged instructions.

2. That the identified sensitive instructions vary in their method of execution when encountered in the user mode. For example, the HALT causes a trap, a RESET causes a noop, and a RTI is executed in a normal manner, i.e. independent of mode.

3. That the I/O page accessing the user program and the MUNIX user's space area are incompatable.

4. That the virtual machine monitor should be completely separate from the job executing on the virtual machine, i.e. not in the same addressing

space, but at the same time have the ability to access and examine in detail the processes' registers, PS, and memory. The virtual machine monitor must also have the capability of controlling the program's execution.

C. MILESTONES

The acheivement and success of the desired goals requires a great amount of complex design, coding, and process interaction. This is especially true if the time span required to attain these goals includes a series of programming teams. A methodical plan must first be initiated and implemented in order to have the end product useable and efficient. Each hierarchical step of the plan must be logically developed, flexible, and modular for good program control and documentation. The following outlines the essential milestones to meet the objectives of a truly virtualized PDP-11/50.

STEP 1 - Develop the basic foundation and necessary tools to support a limited virtual machine. The virtual machine would execute standalone processes subject to the following restrictions. First, interrupt handling would not be included. Secondly, methods of accessing the I/O page would be restricted. Finally, only those sensitive instructions that are required for basic programming be simulated.

STEP 2 - Develop and incorporate a complete interrupt handling process.

STEP 3 - Complete the simulation of all sensitive instructions.

STEP 4 - Develop and include an efficient but unlimited I/O capability. The consolidated virtual machine monitor would now have the ability for supporting a virtual machine that could process any stand-alone job.

STEP 5 - Complete the final enchancements of having the virtual machine capable of supporting different operating systems.

Examination of several virtualization projects at other institutions, in the area of man-hour development of a project of this size, estimates this effort to be approximately one to two years depending on the implementation method. Our goal is to complete the first milestone, putting emphasis on developing the tools, plus program modularity for ease of future enchancements.

D. POSSIBLE APPROACHES

To meet the required virtualization objectives of the school, four possible approaches were examined. The approaches varied considerably in cost, sophisication, and implementation ease. The approaches were:

1. Acquire if possible, virtual machine monitor software already developed for the PDP-11 computer family from anotner institution or source. This aoproach is the most logical and has the greatest appeal in first, not 'reinventing the wheel', and secondly, spending the majority programming effort on converting and adapting it to our environment.

2. Acquire and implement necessary hardware modifications to the PDP-11 system so as to improve its virtualizable characteristics. A true and efficient virtual system can never be realized on a PDP-11 computer until the execution of all sensitive instructions in the user mode automatically interrupt the system's processing. This type of modification of the hardware though, may lead to many undesirable side effects. For example, vendor maintenance agreements may be endangered, plus the real possibility of degradation of normal processing.

3. All programs to be executed on the virtual machine would first be examined by a preprocessor. The function of the preprocessor would be to identify and substitute a processor trap instruction for all sensitive instructions encountered.

This approach has the advantages of being the most cost sensitive and relatively easy to

implement but has three main disadvantages. First, the time spend for preprocessing; secondly, the ease of anyone intentionally or unintentionally beating the preprocessor since data and instructions are interspersed in a program and confusion between the two are easily made; finally, and most important, the inability of processing a program which contains self-modifying code. This includes many operating systems.

4. Implement some extensive operating system modifications to ease the virtualization effort.

Although this approach is in variance with the restriction on minimizing operating system modifications, some of the major problems could be resolved. For instance, it would be highly desireable to have the I/O page available to the virtual processes thus solving the I/O problem and either eliminate or relocate the user stacks. This would require the operating system to distinguish a real from a virtual process for appropriate stack handling and memory mapping.

Another very appealing implementation method is to have the virtual machine monitor execute in the supervisor mode while the virtual processes run in the user mode. This would solve the memory address space problem and have the virtual monitor above the user but less then the operating

system. This also would require extensive modifications since MUNIX only uses the kernal and user mode.

There are always considerable techinical problems in modifying any operating system. This is especially true at the present time of the UNIX system. It is first, relatively new to all personnel at the school, including the C-language which it is written in, but more important the operating system is almost completely undocumented. The majority of effort to date has been in learning, documenting, and modifying the more important areas of the system.

V. SELECTION AND IMPLEMENTATION

A. SELECTION

The preprocessor approach was selected as the method for virtualizing the PDP-11/50. Although this method has some substantial disadvantages as outlined in chapter IV, the cost, ease of implementation, and time limitations more than compensated for them. This method also assisted in overcoming some of the conflicts and undesireable effects of running under MUNIX and without making any modifications to the MUNIX system.

The search for existing virtual machine monitor software, for the PDP-11/50, identified only one other educational institution involved in this area. The University of California, at Los Angeles, under the direction of Professor Gerald J. Popek is currently involved in virtualizing a PDP-11/45 computer. One of their main research efforts to date has been in the area of computer security and the virtual machine. Their implementation method involves a type I virtual machine monitor plus an additional plugable hardware modification module, (the Virtual Machine Extension - KBS11-A), designed and manufactured for them by Digital Equipment Coporation (DEC). The hardware module simply plugs into the system when the virtual machine monitor is running and efficiently traps all sensitive instructions.

Professor Popek was contacted for the possibility of using any of their software modules in our virtual machine

monitor. It was decided that at this time, it would be very impractical since their virtual monitor is still in the development stage, written in a different language, and mostly undocumented. Professor Popek did recommend though, that we purchase from DEC the hardware 'virtual machine extension'. The virtual machine extensions option (KBS11-A) is an available Digital Equipment Corporation option for the PDP-11/45 and PDP-11/50 computers at a cost of $5995. Its specific function is to facilitate the writing of virtual machine monitor code.

The KBS11-A provides the following features: [10]

"1. It provides the ability to trap certain "control- sensitive" instructions when they are encountered in user and/or supervisor mode. These instructions, characterized by the fact that they operate differently in user or supervisor mode than in kernel mode, are:

HALT - Halt Instruction Execution
WAIT - Wait for Interrupt
RESET - Reset the system
RTI - Return form Interrupt
RTT - Return from Trap
SPL - Set Priority Level
MTPI - Move to Previous Instruction Space
MTPD - Move to Previous Data Space
MFPI - Move from Previous Instruction Space
MFPD - Move from Previous Data Space

In the virtual machine environment, these instructions must be trapped and then simulated by the operating software. To aid in the servicing of these instructions, the KBS11-A hardware encodes each of these instructions into a unique 4-bit number which can be read by the service routine. This eliminates software decoding of the instructions. The specified instructions cause a reserved instruction trap, vector 10.

2. It provides the ability to set up an alternate vector space to be used for traps and interrupts from user mode. The effect is to "offset", by 1000, 2000, or 3000 (octal), any vector address generated while the processor is in

user mode. This is done so that a supervisor program can service certain traps from user programs directly while still allowing the Kernel mode program full control of the system (all vectors are in Kernel address space).

3. It provides a User Stack Limit Register, similar in operation to the Kernel Stack Limit Register. Without this capability, the only other way of limiting the user stack is by means of the Memory Management unit, which limits virtual machine structure and flexibility."

The KBS11-A option contains no operator controls, but is under full program control by means of three registers on the UNIBUS. The three registers are the following:

1. VCSR VMX Control-Status Register - This register contains seven control bits to enable or disable the various extended processor features.

2. VCODE VMX Reserved Instruction Code Register - This register contains a four bit code for easy identification of the instruction that caused the trap.

3. VUSL VMX User Stack Limit Register - This register contains the user stack limit address which operates like the standard kernel mode stack limit register.

When the KBS11-A is disabled, the KBS11-A hardware has no effect and the processor operates as normal. This hardware option solves all of the hardware problems and lends itself for creating a true virtual machine. The cost of the KBS11-A is currently the prohibitive feature, but

until this hardware option or one similar to it is installed the goals of the school toward virtualization will never be attended.

B. DESIGN CONSIDERATIONS

With the objective of implementing a basic virtualized PDP-11/50 computer (chapter IV - first milestone), the following design characteristics and considerations were adopted.

1. The virtual machine's configuration will consist of one PDP-11/50 computer without a memory management unit, 32k words of memory, and one LA30 DECwriter termininal.

2. Job processing will be restricted to standalone jobs that have been preprocessed for sensitive instruction substitution. Jobs also will be restricted in the method of accessing I/O and in not using interrupt type processing. The 'MOV' instruction employing direct indexing will be the only instruction allowed to reference the I/O page and the 'HALT' instruction will be used for exiting from the user program.

3. The debugging aid will include the following capabilities: (1) Be able to display the contents of the general registers, program status word and memory. Memory may be displayed both

octally and symbolically; (2) Be able to modify any register or memory cell; (3) Provide the ability for breakpoint processing.

4. The program will be written in the C-language, be modular in design for ease of incorporating enhancements and understanding, and incorporate a built in internal debugger that will upon command, give the value of the variables encountered throughout the program.

C. IMPLEMENTATION

1. Program Overview

The virtual machine monitor consists of the following functional units. (Figure 11 illustrates these functional units and their relationships).

a. Supervisor Monitor

The supervisor monitor is the highest level module of the virtual monitor. It provides the main communication between user and virtual machine. It accepts and edits the virtual machine monitor commands and calls and supervises their execution.

b. Preprocessing Monitor

The preprocessing monitor edits, calls, and monitors the preprocessor executions.

c. Preprocessor

The preprocessor is a completely self-contained module that examines incoming virtual machine job for

PROGRAM MODULE OVERVIEW
VIRTUAL MACHINE MONITOR

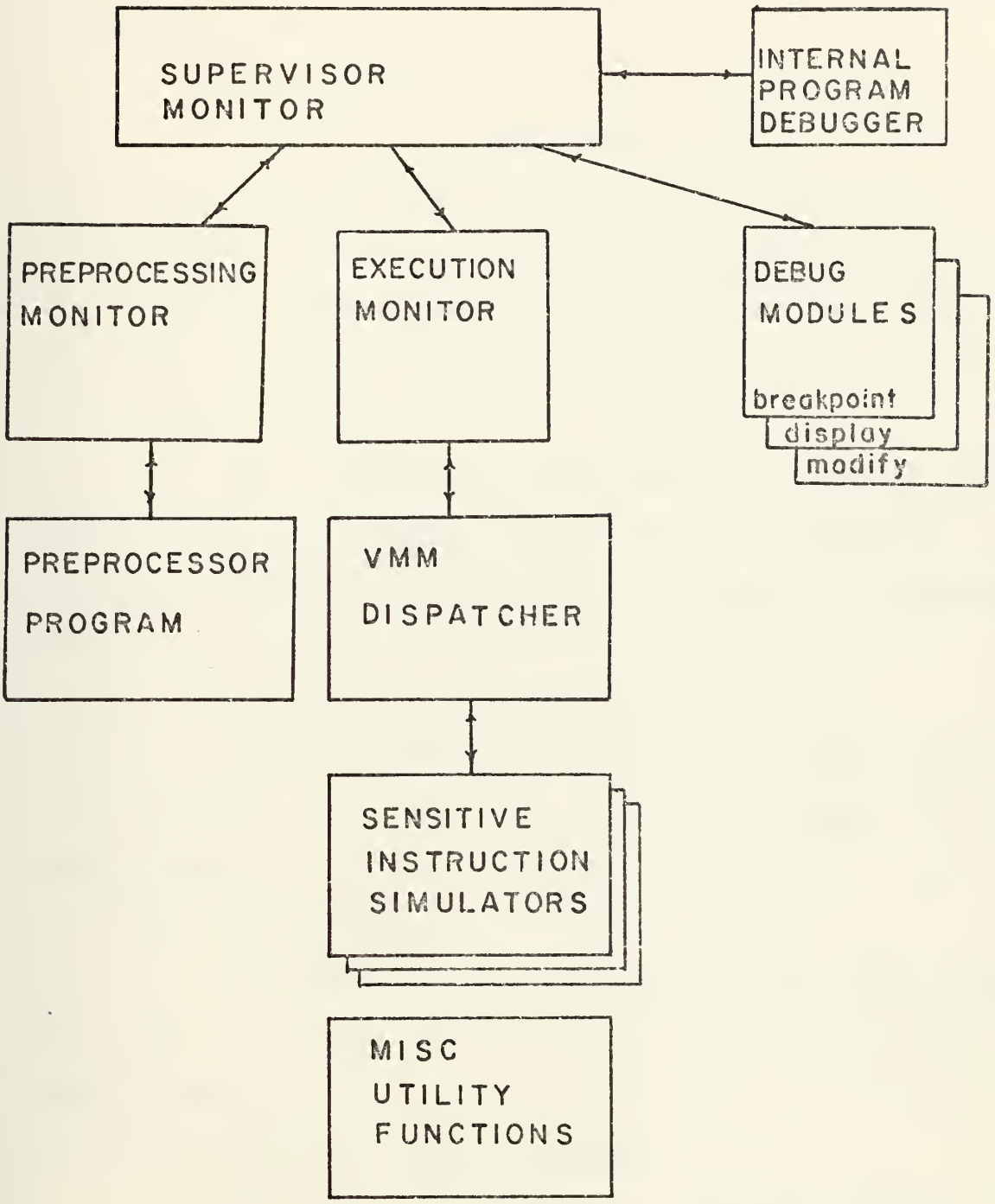


FIGURE II

senstitive instructions and possible substitution. Its output is a modified executable (a.out) file and a sensitive instruction table.

d. Execution Monitor

The execution monitor's function is to start or continue the executing of the job on the virtual machine. If a breakpoint or sensitive trap is encountered in the program the execution monitor receives the interrupt and calls the dispatcher for servicing.

e. Dispatcher

The dispatcher determines the type of problem encountered in the program and calls (if necessary) the required sensitive instruction simulator or breakpoint handler.

f. Sensitive Instruction Simulators

These routines (one for each sensitive instruction) simulate the execution of the individual sensitive instructions.

g. Debug Modules

The debug modules are available to the general user for assistance in program debugging and in examining program execution.

h. Internal Program Debugger

When this is activated, variable values used throughout the program are displayed to aid in debugging. This function is not intended for the general user but instead for the virtual machine's monitor maintenance, debugging, and enchancement verification.

i. Miscellaneous Utility Functions

This is a general collection of utility functions used by the main modules.

2. Supervisor Monitor

The supervisor monitor (named 'evmm') is a small simple routine which controls and directs the main flow of the program, as directed by the user's commands. It is the highest level module of the virtual machine monitor and the one that is entered when the monitor program is executed. Its action is simply to wait for a user command, edit the command, and initiate a call to the correct module for processing. After the called process has completed, control is returned to the supervisor monitor. The supervisor monitor then waits for the next command.

The following is a list of the valid commands and their respective functions.

COMMAND	FUNCTION
p filex [y/n] [b]	Preprocess filex. This command has two optional parameters. The first indicates whether the user wants to be informed when a sensitive instruction is encountered. The second activates the preprocessor's internal debugger.
e	Start or continue program

	execution.
b ADDR	Put a breakpoint at address ADDR
d r	Display all general registers plus the program status word.
d o ADDR #	Display memory in octal starting from address ADDR for # number of words.
d s ADDR #	Display the program symbolically from address ADDR for # number of instructions.
m rX VALUE	Modify register X to VALUE (octal)
m SP VALUE	Modify stack pointer to VALUE (octal).
m PC ADDR	Modify the program counter to address ADDR.
m PS VALUE	Modify the program status word to VALUE (octal).
m mem ADDR VALUE	Modify memory at address ADDR to VALUE (octal).
z	Enter internal debug.

x	Leave internal debug.
s	Terminate the execution of the virtual machine monitor.

3. Preprocessor

The preprocessor program (named '←vmm001') is a self contained module whose function is normally required only once for each job processed on the virtual machine. Because of its infrequent use, the preprocessor is loaded into memory and executed only when needed. The preprocessor is called by the preprocessing monitor. The interaction between the preprocessing monitor and the preprocessor program is coordinated by a series of MUNIX system calls as illustrated in figure 12 .

The input to the preprocessor is programs which are to be executed on the virtual machine. These programs must be in the form of an executable (a.out) file with non-sharable segments. The user program input is read into the preprocessor in blocks of 512 bytes for improved effeciency. The preprocessor program examines and tests each program instruction in the text segment for sensitivity. If such an instruction is found the 'TRAP' instruction (104450) replaces it and the sensitive instruction is then written in the sensitive instruction table.

The preprocessor program has three arguments. The first argument is mandatory and gives the name of the program (a.out file) for preprocessing. The second argument is

INTERACTION BETWEEN PREPROCESSING MONITOR AND PREPROCESSOR

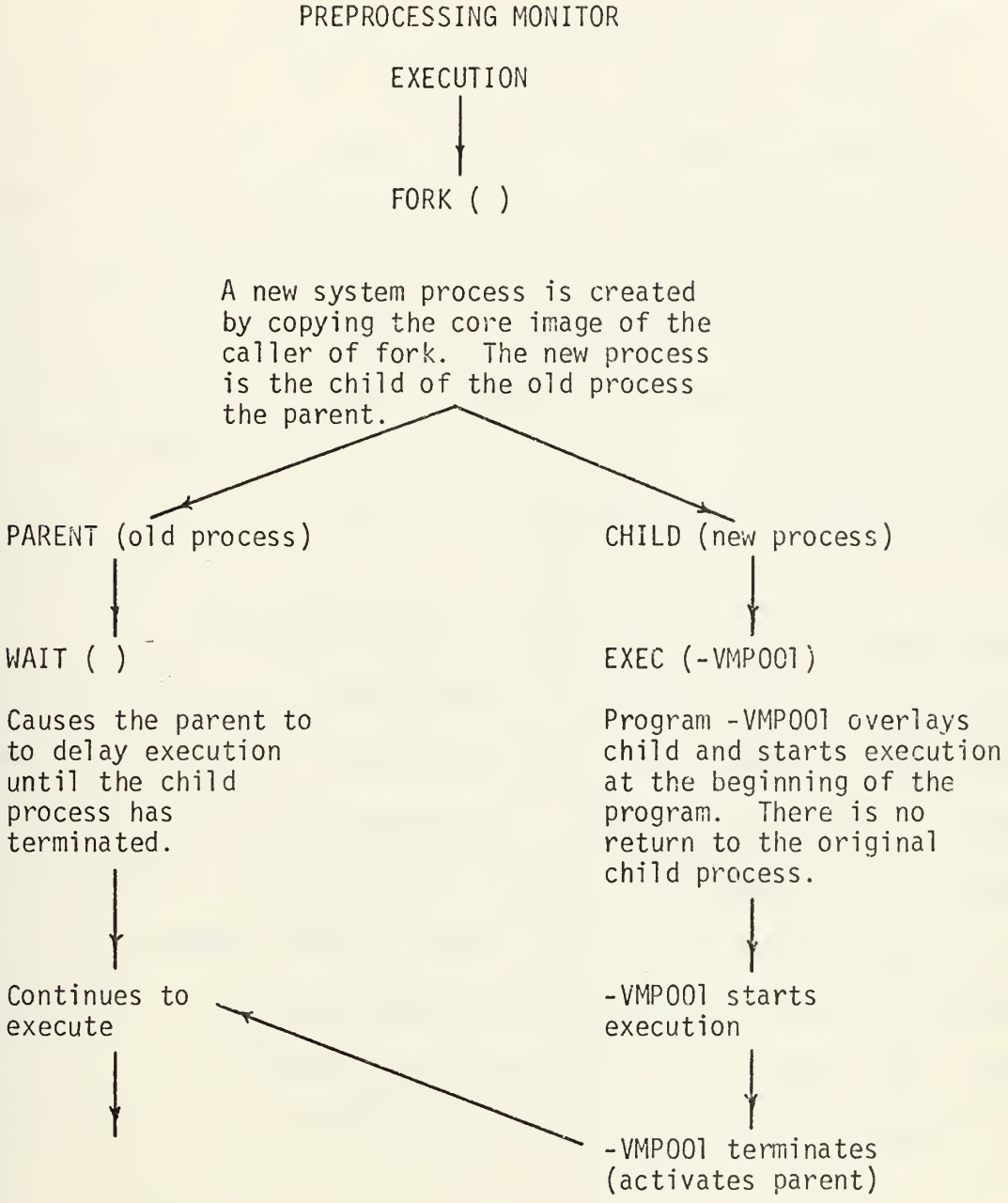


FIGURE 12

optional allowing the user a choice of whether he wants to be notified when a sensitive instruction is encountered. (Default value is 'do not notify'). When notification is desired, the sensitive instruction plus the five previous instructions of the program and their location are displayed. It is at this time that the user can exercise the option of agreeing or disagreeing. If the user agrees, the 'TRAP' instruction will be substituted, otherwise no substitution will be made. It must be emphasized that in the notification mode, the user decides if a sensitive instruction has been encountered. The final optional argument is the internal debug switch for the preprocessor (Default value 'is no debug').

The 'JSR' and 'TRAP' instructions have the possibility of having a variable number of parameters following them (these parameters frequently look like 'JSR' or 'TRAP' instructions to the preprocessor). Because of this unique characteristic, the preprocessor will automatically printout the five previous instructions, the sensitive instruction encountered, and a message requesting the number of parameters that are following the instruction. The user can then indicate to the preprocessor the number of parameters to skip.

The preprocessor's output is a modified a.out file (named ←vmm002) void of all sensitive instructions and a sensitive instruction table (named ←vmm003) containing the encountered sensitive instructions and their addresses.

The program logic flow is illustrated in figure 13.

PREPROCESSOR

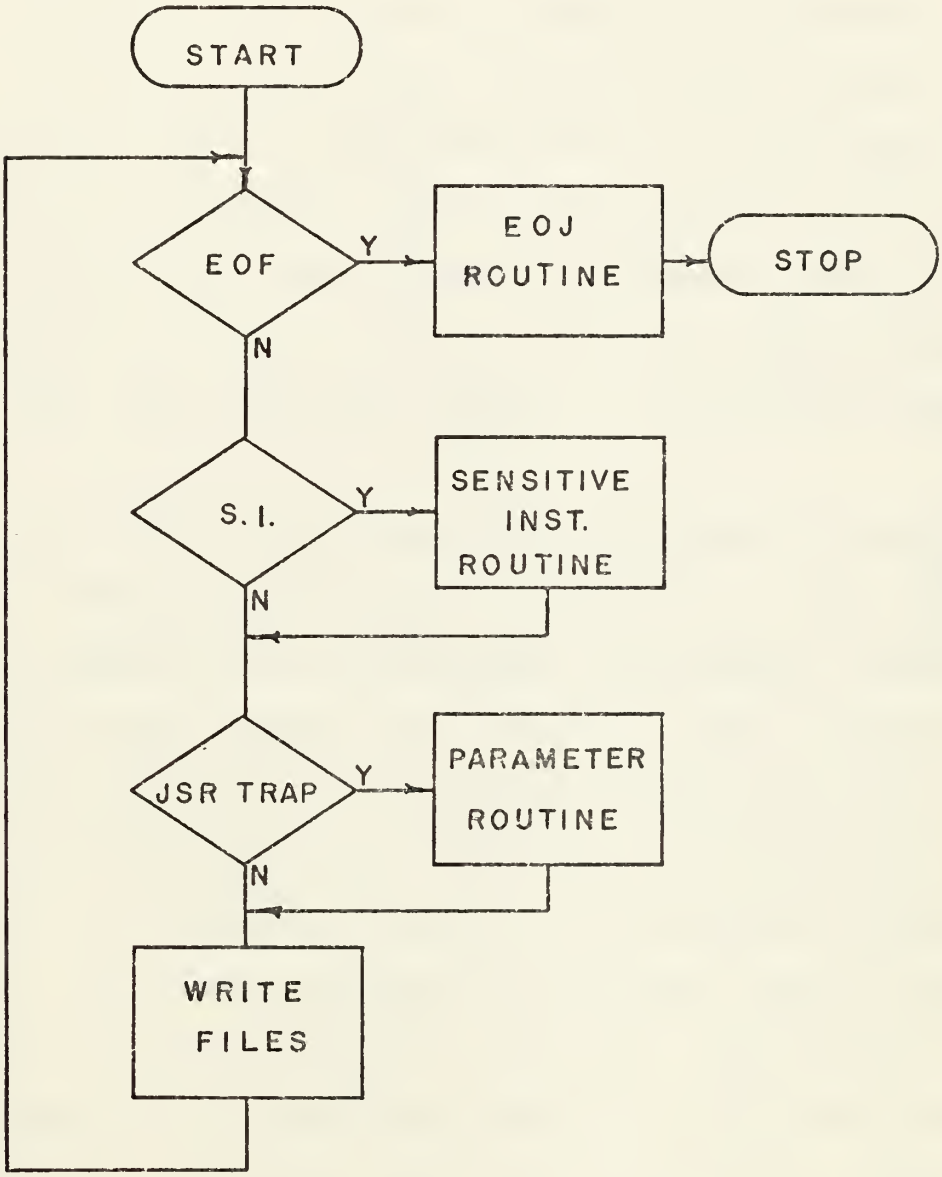


FIGURE 13

4. Execution Monitor

The execution monitor controls the starting and restarting of processes executing on the virtual machine. All processes running under MUNIX have two executable forms. The first form is the a.out file for programs ready to be executed. The second form is the core image file used during execution (figure 14). The virtual machine monitor frequently accesses one or both of them (depending on if the process has been initially executed). During the course of program execution these two files differ in location. The a.out file is located in the user's library while the core image file is, when not executing, located in the system's swap area.

If the process has never been executed, the execution monitor initiates the process' execution in the same manner (fork, exec) as the preprocessor was executed (figure 12). After the user program has initially been executed, the execution monitor need only issue a CONTIN system call. This system call puts the core image file, which is currently blocked for processing, back on the system's process ready queue. The execution monitor then issues another WAIT system call to wait for the next interrupt. The execution monitor returns to the supervisor monitor only when the job is completed or a breakpoint has been encountered. Figure 15 illustrates the execution monitor's program logic.

5. Dispatcher

The dispatcher is the main control module for processing virtual machine interrupts. The dispatcher is

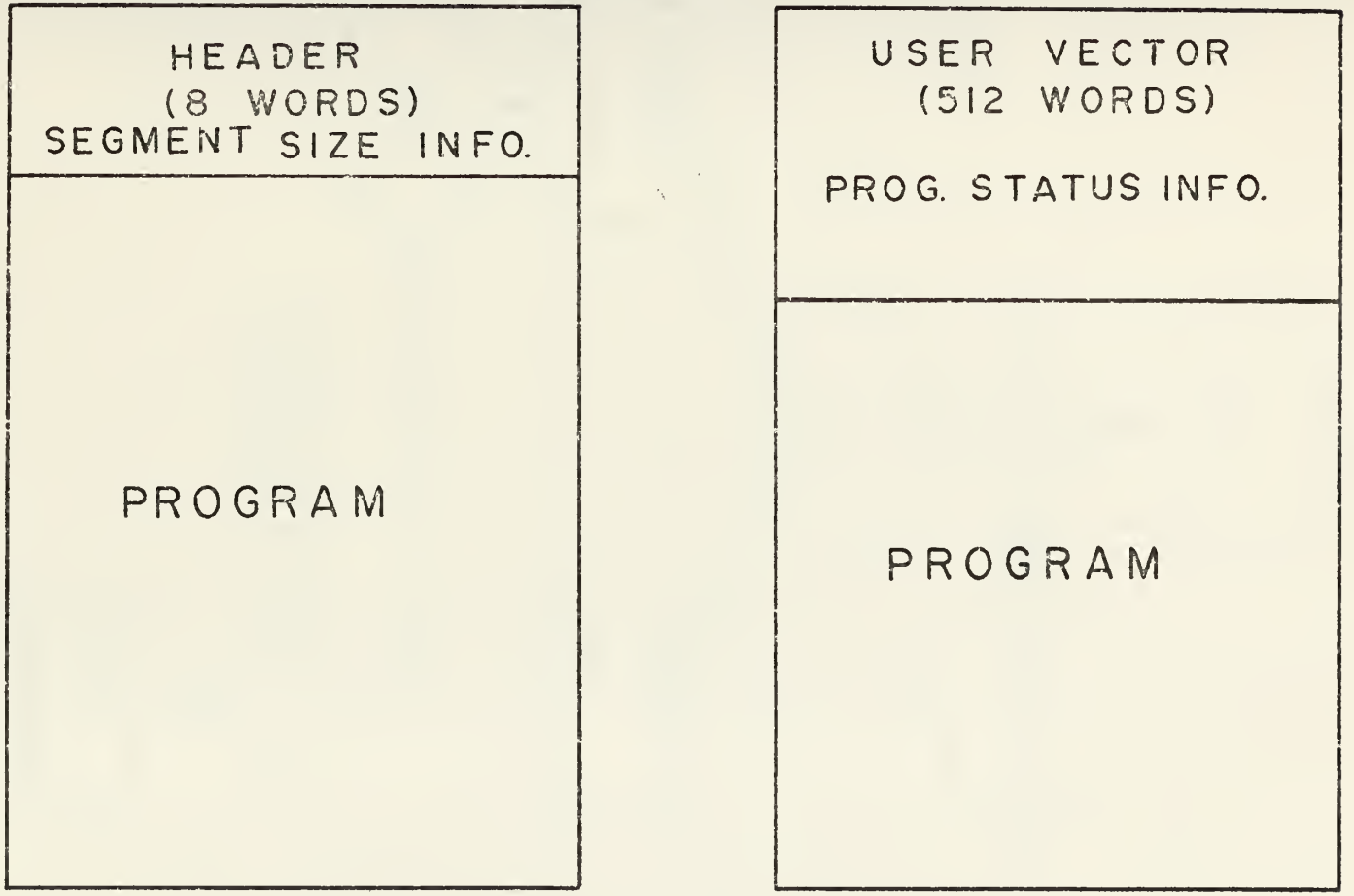

HEADER
(8 WORDS)
SEGMENT SIZE INFO.
PROGRAM
A.OUT FILE
BEFORE EXECUTION
USER VECTOR
(512 WORDS)
PROG. STATUS INFO.
PROGRAM
CORE IMAGE FILE
AFTER EXECUTION

FIGURE 14

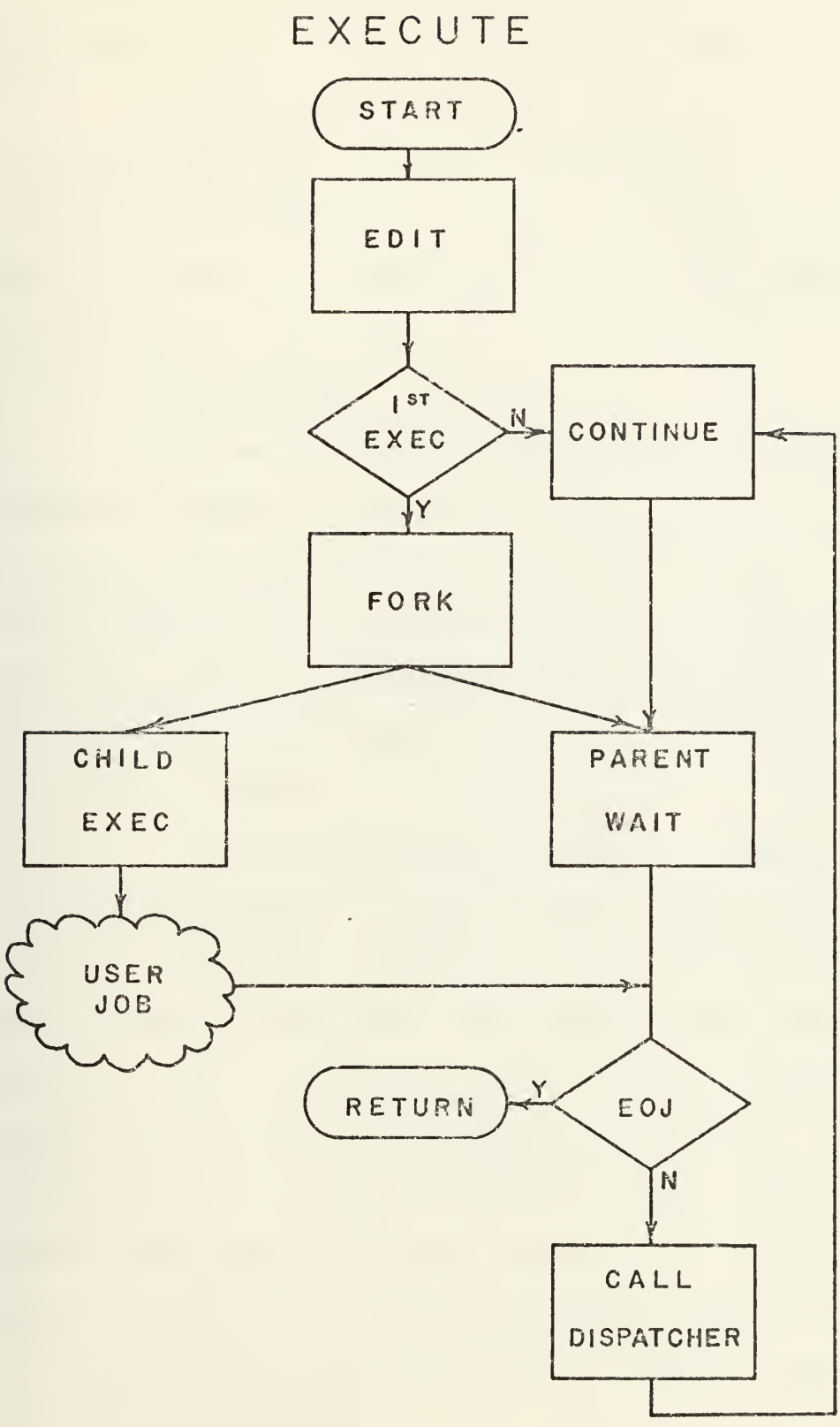

EXECUTE
START
EDIT
1ST EXEC
N
CONTINUE
Y
FORK
CHILD EXEC
PARENT WAIT
USER JOB
RETURN
Y
EOJ
N
CALL DISPATCHER

FIGURE 15

involked when the program executing on the virtual machine executes one of the substituted 'TRAP' instructions.

The dispatcher determines where in the program the interrupt occurred, the type of interrupt, (i.e. breakpoint, sensitive instruction, or normal interrupt) and finally, what simulator or function to call. The program logic is illustrated in figure 16.

6. Debug Modules

Plans were originally made to use the debugging programming modules of UNIX, but unfortunately, the debugging program modules were being completely revised to be used under MUNIX so the decision was made to write the virtual machine monitor's own debugger. The three primary modules of the debugger are the display, modify, and breakpoint.

a. Display

The display module is used to display the virtual machine's register values or the virtual machine's memory content. (Display commands are outlined in Appendix D.) Upon command of the user, the values of the general purpose registers are displayed both in octal and decimal number representation while the values of the PS, SP, and PC are displayed only in octal. Memory may be displayed in octal (core dump format) or symbolically. In either case, the memory display will start from where requested by the user for the number of words specified. Module logic flow is depicted in figure 17.

The core image file is used to retrieve the values of the registers and the octal content of memory.

DISPATCHER

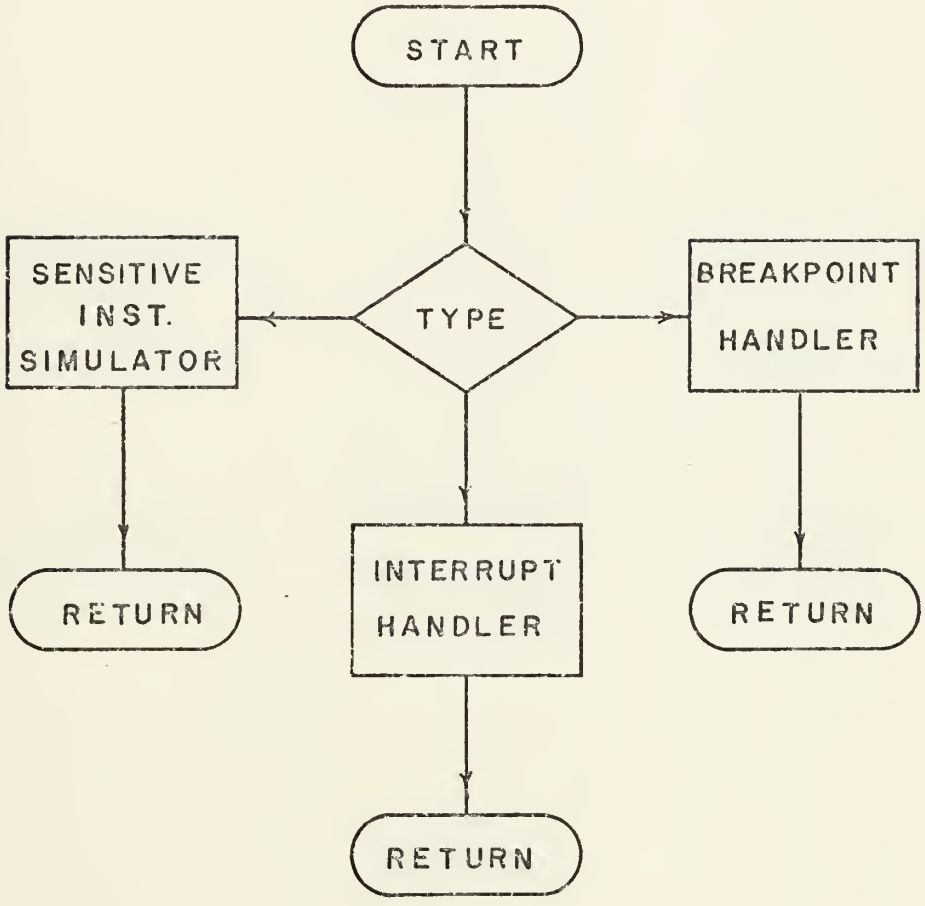


FIGURE 16

DISPLAY

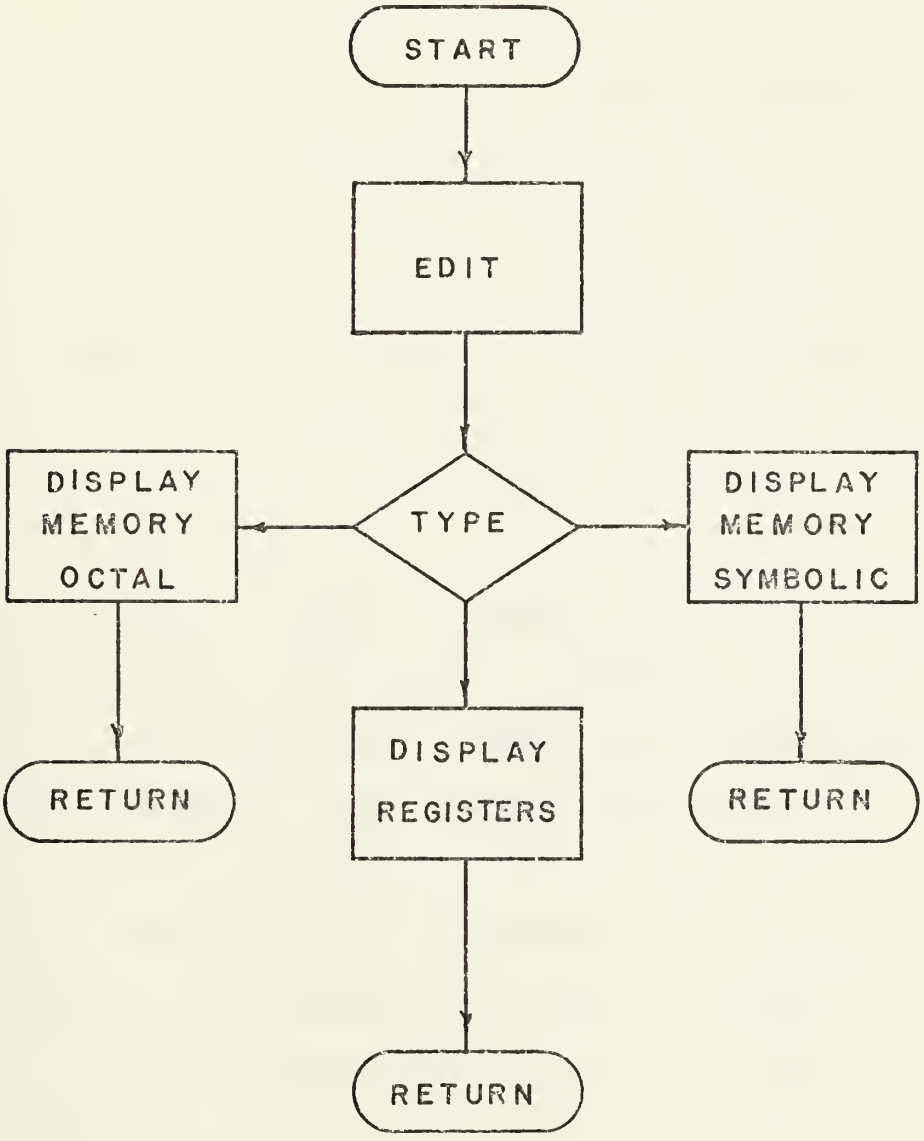


FIGURE 17

Thus the program must first be executed (produces a core image file) before the use of these two commands are allowed.

b. Modify

The logic flow of the modify module (figure 18) is similar to the logic of the previously discussed display module. Memory is modified by the modify module on a word-by-word basis. As was in display module, it is also the case here that the program must first be executed before a register is modified. Any memory modification (memory may be modified before execution) will automatically modify the a.out file and the core image file if it exists.

c. Breakpoint

The breakpoint debugging facility consists of two basic functions. One function (BPINSERT) inserts the new breakpoint into the user program and the second function removes the breakpoint when executed (BPHANDLER).

Up to ten breakpoints at one time can be inserted into the program and can be entered either before or during execution (For breakpoint commands see Appendix D).

The BPINSERT function saves the instruction at the breakpoint address in an array and inserts a 'TRAP' (104450) instruction in place of it.

When the breakpoint is executed during user program execution an interrupt occurs and is serviced by the BPHANDLR (via the dispatcher). The real instruction is returned to the program and the program counter (PC) is backed-up one instruction to point to the returned

MODIFY

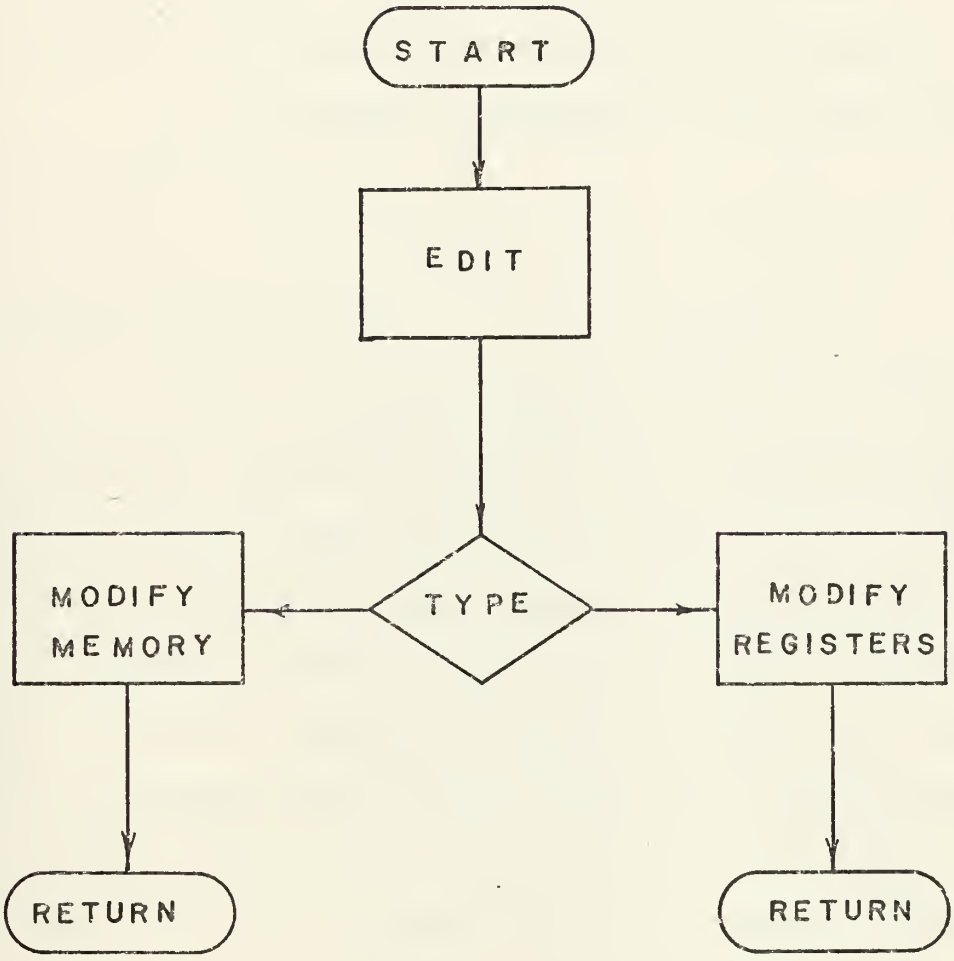


FIGURE 18

instruction. A message is printed to the user showing the instruction address and instruction where the breakpoint was inserted. Program control is then returned to the supervisor monitor.

Breakpoints are removed upon execution and must be re-inserted upon the completion of a breakpoint interrupt each time the user desires the breakpoint to remain.

The BPINSERT and BPHANDLR logic flow are illustrated in figure 19.

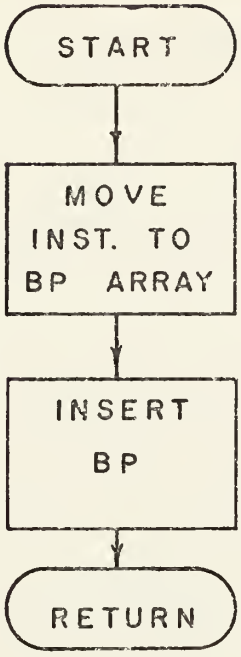
D. TEST AND EVALUATION

Several small standalone programs were written to test the reliability and performance of the virtual machine. The testing procedure included executing the program on the virtual machine and on the bare machine using the same input.

All programs tested on the virtual machine executed in a similar fashion producing the same output as in the standalone environment, but with a substantial and disappointing lost of efficiency. For example, a small program designed to echo print a string of characters had immediate response on the bare machine, but took up to twenty seconds per character on the virtual machine.

The virtual machine's lack of efficiency is mainly caused by the need to simulate I/O to a real device, the user program waiting for execution in the process queue after the virtual machine monitor has completed its function, and the need for updating the super-block via the SYNC

BPINSERT



BPHANDLER

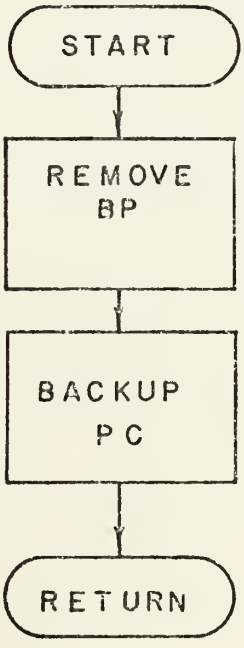


FIGURE 19

system call (causes all information in core memory that should be on disk to be written out) to insure the user core file is located on the disk 'swap' area before the virtual machine monitor can access the program.

The efficiency aspect makes the virtual machine under its current design, highly impractical for serious utilization, and emphasizes the need for a hardware virtual machine extension.

VI. CONCLUSION

The virtual machine with its numerous applications provides many research opportunities for an educational institution. The virtual machine has proven itself as an extremely valuable tool in the university computer laboratory, in the computer science classroom as an effective training aid, and in the business world as a sophisticated approach in solving difficult application problems.

Although extremely limited in its current state, the PDP-11/50 virtual machine has arrived at the Naval Postgraduate School. The first milestone has been accomplished. The basic foundation for a useable virtual machine with accompanying debugging aids and tools has been completed. The virtual machine will execute stand-alone processes that comform to the stated restrictions. Nevertheless, a great deal remains to be accomplished to produce the type of virtual machine desired and needed at the Naval Postgraduate School.

In our study, design, and implementation of this limited virtual machine many conclusions, afterthoughts, and recommendations have come to light. The following highlights some of our main conclusions and thoughts.

1. Although we have only developed the embryo of the virtual machine, its usefulness is readily apparent. For instance, an inexperienced computer science student can design a simple assembler program and actually see and follow the program's

execution by examining the program status, memory, and registers. He can rapidly obtain an understanding of data representation versus instruction representation, symbolic language versus machine language, and memory and CPU interaction. Basic I/O methods can also be explored.

2. Being ambitious is an excellent personal trait, but trying to complete a new house with a fancy and beautiful exterior before the foundation is completed, is catastrophic. A great deal of time and effort should be devoted to dividing the problem into logical pieces of workable sizes. There was a strong desire on our part to complete and implement a virtual machine that satisfied the entire school's goals for virtualization. At first, much time was spent in designing some of the more intricate and final enhancements. It was with great effort and some loss of pride that we limited our goals and establish a milestone approach.

3. Being the first to work on brand new equipment has a psychological fulfillment, but this joy is quickly diminished on encountering the ever present new hardware bugs. In the planning phase of new applications that will be implemented on new hardware (not necessarily unfamilar hardware) 'safety' factors should be incorporated

in the schedule. At worst you are prepared and at best, you finish ahead of schedule.

4. Like new equipment, unfamiliarity with the language adds a new dimension of problems. There is a substantial learning before one can be productive. This takes time, effort, and continuous self-discipline, but is absolutely essential.

5. Many enhancements on the PDP-11/50 were being accomplished concurrently in a coordinated team-group aproach. A marriage of advantages exists in this type of real world working environment. Few things can be substituted for actual experience in a hands-on integrated approach. Design considerations must be synchronized with the other teams, working schedules must be coordinated, and joint planning meetings must regularily be attended. On the other hand, one is continually frustrated by files being destroyed, system crashes, unknown system changes, and lack of detailed coordination.

6. The program design has changed considerably from the start. It is extremely difficult to start by developing programming standards, especially when the language and its characteristics are unfamiliar. Much time was spent on reviewing the program for possible regrouping and

consolidating program modules.

The first modules were generally poorly written, caused mainly by language unfamiliarity and the desire to have the program just work. As programming skills improved, the program modules tended to become more sophiscated and clever, and resulted in the disguise of the logic flow. This soon 'backfires' on any program of considerable size. Special care and time was finally taken to improve program modularity, clarity, and simplicity in program logic, especially when it was envisioned that later enhancements would be added to the program by other programmers.

The use of variables was another problem area. They must be controlled from the beginning or it will quickly be too late. Variables should be organized to readily distinguish their purpose and should incorporate some type of naming conventions for better program control and maintenance.

One good feature of the program, incorporation of an internal programming 'debugger' actuated on command, has proven itself extremely useful and valuable.

As previously mentioned, there is no 'one best way' to design a virtual machine, however access to various developed ideas and alternatives is a catalyst for futher research.

As noted throughout this thesis, there is a variety of ongoing activity in progress throughout the country concerning the virtual machine approach, but much more work and educated professionals are needed. The future of the virtual machine at the Naval Postgraduate School will not reach an acceptable level without careful attention. Many areas remain unresolved including the possible acquiring of the DEC virtual machine extension option and substantially modifying the operating system to alleviate the I/O problem.

Numerous areas of research in virtual machines could be recommended for futher research, but until a more versatile virtual machine is implemented these arts will have to wait or be accomplished at another institution.

APPENDEX A

GLOSSARY

The purpose of the glossary is to provide a brief list of the more common definitions found in this thesis and related literature.

1. ARCHITECTURE - Attributes of the system as seen by the user. It is the functional behavior and structure of the system where emphasis is placed on the needs of the user.
2. BARE MACHINE - Machine without its accompanied software.
3. CONTROL PROGRAM - Generally means virtual machine monitor.
4. EXTENDED MACHINE - The bare machine plus operating system.
5. EMULATION - Technique to map one system architecture into another different architecture using some type of firmware support. Enables the computer to execute instructions of other computers.
6. FAMILY - Series of computers with similiar architectural design and compatabilities, i.e. IBM 360 family.
7. FAMILY VIRTUALIZING - Virtual machine not identical to the host but is a member of the same computer family.
8. GENERATION - Refers to the architectural characteristics of the computer, i.e. second generation and third generation computers.
9. HARDWARE VIRTUALIZER - Hardware-firmware virtual machine monitor that supports a virtual machine, i.e. distinct from software design to support a virtual machine.

10. HYBRID VIRTUAL MACHINE - All instructions in the privelaged state are software interpreted. All others are executed directly.

11. HOST MACHINE - Bare machine in which VMM runs.

12. NATIVE MODE - Operation of system, i.e. no emulation.

13. PRIVILEGED INSTRUCTION - Instruction that must execute only in the privileged mode for correct results.

14. SELF-VIRTUALIZING - The processor of the virtual computer is is identical to the host.

15. SENSITIVE INSTRUCTION - An instruction which if permitted to execute directly on the host machine will give incorrect results because of the virtual machine software construction.

16. SUPERVISOR CALL - Instruction used to call upon the operating system.

17. VIRTUAL MACHINE - Isolated duplicate of a real machine.

18. VIRTUAL MACHINE MONITOR - Software which mediates between the virtual machine and host.

19. VIRTUAL MACHINE MONITOR TYPE I - Runs on a bare machine.

20. VIRTUAL MACHINE MONITOR TYPE II - Runs under the host machine's operating system.

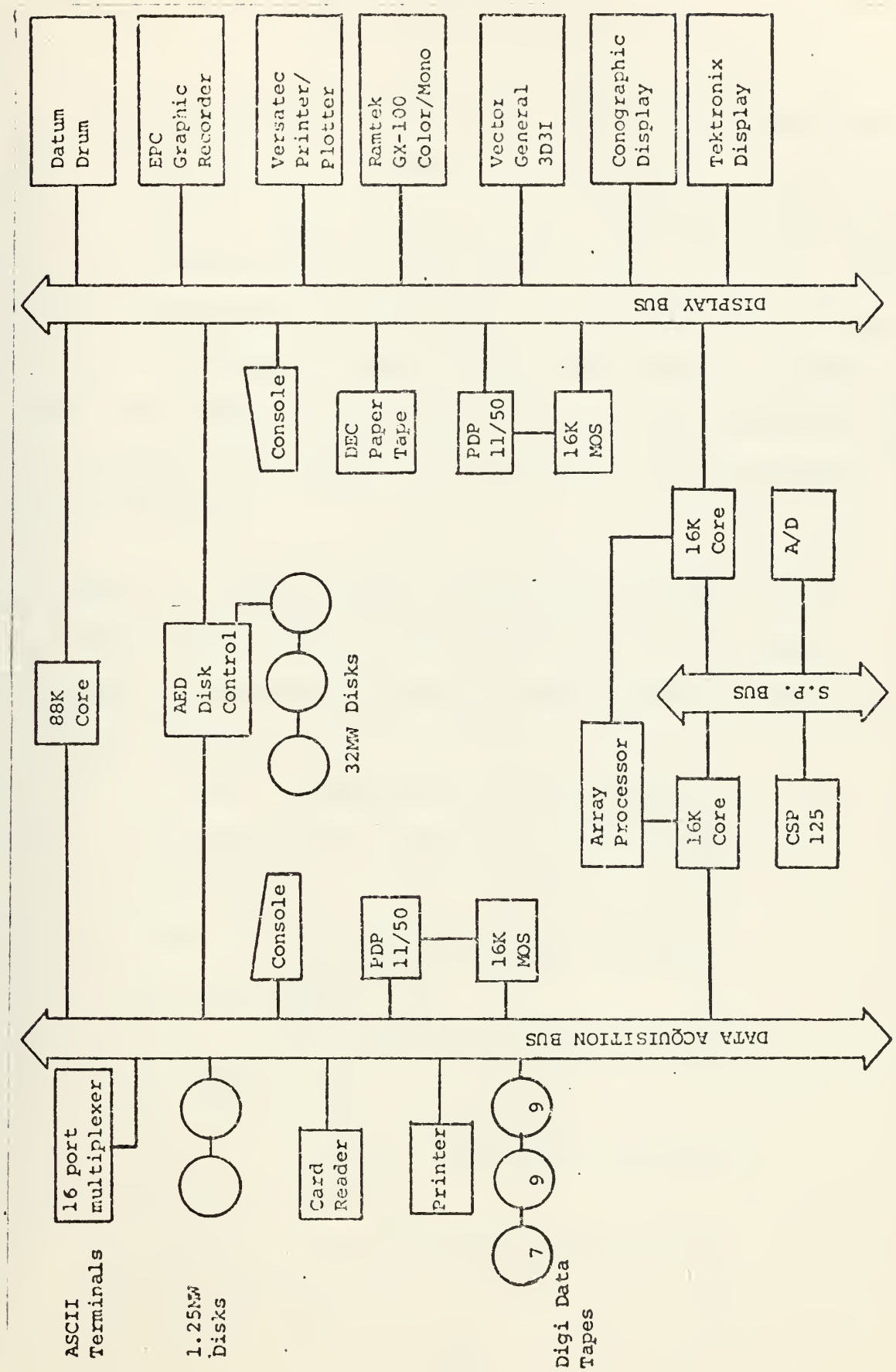
21. VIRTUALIZATION - The creating of a virtual machine.

22. VIRTUALIZABLE - Refers to being able to create a virtual machine from the existing machine.

23. VIRTUAL MACHINE RECURSION - Ability to run a virtual machine monitor on a virtual machine.

APPENDIX B

PDP-11/50 CONFIGURATION



APPENDIX C

SENSITIVE INSTRUCTIONS

The following PDP-11/50 machine instructions were determined sensitive. The determination was based on the premise that if the instruction were allowed to be executed either (1) the current state of the virtual machine would be inaccurate and/or (2) that the instruction would unitentionally interfer with and/or return to the real system. It should be noted that some of these instructions are only sensitive due to the approach selected for producing and operating the virtual machine.

GROUP 1. - These are the instructions that if executed would trap to the real system for interrupt handling instead of to the virtual machine interrupt handlers.

1. HALT - Generates an interrupt in user/supervisor mode, traps to physical address 4.
2. BPT - Breakpoint interupt, traps to physical address 14.
3. IOT - I/O interrupt, traps to physical address 20.
4. EMT - Emulator interrupt, traps to physical address 30.
5. TRAP - Trap interrupt, traps to physical address 34.

GROUP II - These are the instructions that if executed would automatically change the program status word (PS). This action must be simulated, in a virtual interupt changing only the virtual PS.

1. RTI - Return from trap.
2. RTT - Return from trap.
3. WAIT - It causes the processor to relinquish use of the UNIX and wait for an external interrupt.

GROUP III - These instructions are designed for inter-mode communication using the memory management unit. A virtualized memory management unit is required for this instruction set.

1. MFPD - Move from previous data space.
2. MTPD - Move to previous data space.
3. MFPI - Move from previous instruction space.
4. MTPI - Move to previous instruction space.

GROUP IV - These instructions are NOPs in the user/supervisor mode and should reflect a change in the state of the virtual machine.

1. RESET - All devices on UNIBUS are reset.
2. SPL - Sets a new priority level in the PS.

GROUP V - Miscellaneous instructions.

1. MOV - This refers to any MOV instruction that references the I/O page. This instruction is sensitive only due to the restriction placed on the I/O processing.

APPENDIX D

USERS MANUAL
FOR THE
PDP-11/50 VIRTUAL MACHINE

I. PURPOSE

This users manual is primarly designed to explain how to execute programs on the virtual machine and how to use the virtual machine's debugging capability.

II. VIRTUAL MACHINE

1. Virtual Machine - What it is

A virtual machine is basically a software program which creates the image and environment of a bare machine (a PDP-11/50 without an operating system). Programs that normally execute on a real bare machine will execute on the virtual machine.

2. Function

The virtual machine has been used in many sophisticated applications in the computer processing world. One of these uses, and the primary one for this machine, is an educational aid for the student in computer science.

Here the student has his own bare PDP-11/50 computer to program and examine the program's execution. The virtual machine will execute stand-alone PDP-11/50 assembler programs that meet the restrictions in section VII.

3. Debugging Capability

The virtual machine has a simple but significant debugging capability. The student can display and modify the general purpose registers, program counter, stack pointer, and program status word. He can also display and modify memory. He can insert breakpoints at certain areas of his program for controlled program execution.

4. Configuration

The virtual machine's configuration is as follows:

a. One PDP-11/50 computer (without memory management).

b. 32k words of memory.

c. One LA30 DECwriter terminal.

III. INPUT TO THE VIRTUAL MACHINE

Jobs to be processed on the virtual machine must conform to the following characteristics. They must:

1. be PDP-11/50 assembly programs.
2. be in a.out file format.
3. have non-shareable program segments.
4. not incorporate any UNIX system calls.
5. have been preprocessed by the virtual machine monitor.

IV. GENERAL FLOW IN USING THE VIRTUAL MACHINE

1. Programmer designs, programs, and assembles his program.
2. User starts up the program '~vmm'.

3. User preprocesses his assembled program using the preprocessor command of the virtual machine.

4. Programmer, may if he desires, insert breakpoints into his program at critical areas.

5. User executes his program using the execute command of the virtual machine.

6. Programmer can at anytime while his job is not actually executing on the virtual machine, utilize any debug function.

7. When program has terminated, user can stop the virtual machine by the termination command.

V. VIRTUAL MACHINE COMMANDS

The following is a list of the valid commands and their respective functions. All addresses (ADDR) and required values (VALUE) must be expressed in octal.

COMMAND	FUNCTION
p filex [y/n]	Preprocess filex. This command has one optional parameter. The parameter indicates whether the user wants to be informed when a sensitive instruction is encountered.
e	Start or continue program execution.
b ADDR	Put a breakpoint at address

	ADDR
d r	Display all general register plus the program status word.
d o ADDR #	Display memory starting from address ADDR for # number of words.
d s ADDR #	Display the program symbolically from address ADDR for # number of instructions.
m rX VALUE	Modify register X to VALUE
m SP VALUE	Modify stack pointer to VALUE.
m PC ADDR	Modify the program counter with address ADDR.
m PS VALUE	Modify the program status word to VALUE.
m mem ADDR VALUE	Modify memory at address ADDR to VALUE.
s	Terminate the execution of the virtual machine monitor.

VI. PREPROCESSOR

The preprocessor examines the user's programs for sensitive instructions (instructions that can not be allowed to

directly execute). As illustrated above in the virtual machine commands, the preprocessor has two parameters. The first parameter is manditory and is the user's program name; the second parameter is optional, allowing the user a choice of whether he wants to be notified when a sensitive instruction is encountered. When notification is desired, the sensitive instruction plus the five previous instructions of the program and their location will be displayed. After examining these instructions the user can agree or disagree with the preprocessor. The user's decision is accepted by the preprocessor so user BEWARE.

The 'JSR' and 'TRAP' instructions have the possibility of having a variable number of parameters following them (these parameters frequently look like sensitive instructions to the preprocessor). Because of this unique characteristic, the preprocessor will automatically printout the five previous instructions, the 'JSR' and 'TRAP' instruction encountered, and a message requesting the number of parameters that are following the instruction. The user can then indicate to the preprocessor the number of parameters to skip.

VII. BASIC RULES AND RESTRICTIONS

1. All files must be an a.out file and be preprocessed before any command will function on the virtual machine including file execution.

2. All files will be stand-alone type processes, and must not use any UNIX system calls (READ, WRITE, EXIT,

ETC.).

3. Files cannot employ any interrupt type processing.

4. Files must perform their own I/O routines. I/O is restricted in the method used to access the I/O page. All access to the I/O page will be performed only by the 'MOV' instruction, and then only by direct indexing. I/O accessing is limited to the four LA30 DECwriter registers. (see DEC peripherals handbook).

5. File must first be executed before displaying or modifying registers, or in displaying memory in octal.

6. Files will use the 'HALT' instruction for normal exiting from their program.

7. The following assembler instructions will be treated as NOP instructions: MFPD, MTPD, MFPI, MTPI, RESET, WAIT, RTI, RTT, and SPL.

8. The user should be cautioned against inserting a breakpoint within an I/O (loop) routine that inputs a string of characters. When a breakpoint is encountered the remaining characters for insertion will be lost.

VIII. FILE NAMES AND FUNCTION

1. '←vmm' - Name of the virtual machine monitor program the initially creates and controls the virtual machine.

2. '←vmp001' - Name of the preprocessor program.

3. '←vmm002' - Name given to the user program after user program is is modified by the preprocessor.

4. '←vmt003' - An array of sensitive instructions and addresses in the user program by the preprocessor.

BIBLIOGRAPHY

1. Bairstow, J. N., "Many From One: The Virtual Machine Arrives", Computer Decisions, p. 29-31, January 1970.

2. Bates, L. A. and Srodawa, R. R., "An Efficient Virtual Machine Implementation," Proc. NCC, AFIPS Press, p. 301-308, 1973.

3. Buzen, J. P. and Gagliardi, U. O., "The Evolution Of Virtual Machine Architecture," Proc. NCC, AFIPS Press, v. 42, p. 291-300, 1973.

4. Buzen, J. P. and Gagliardi, U. O., "Introduction To Virtual Machines," Honeywell Computer Journal, v. 7, no. 4, 1973.

5. Bisbey, R. L. and Popek G. J., "Encapsulation: An Approach To Operating System Security," USC/Information Science Institue, Marina Del Rey, California, October 1973, revised July 1974.

6. Denning, P. J., "Third Generation Computer Systems," Computer Surveys, v. 3, no. 4, December 1971.

7. Digital Equipment Corporation, PDP-11 Peripherals Handbook, 1973.

8. Digital Equipment Corporation, PDP-11/45 Processor Handbook, 1974.

9. DEC PDP-11 Series, M11-384-301, Datapro Research Corporation, Delran, New Jersey, September 1973.

10. Digital Equipment Corporation, "Virtual Machine Research To The PDP-11/45 KBS11-A," CSS-MO-F-9.2-9,1975.

11. Goldberg, R. P., "Survey Of Virtual Machine Research," Computer, v. 12, no. 3, p. 341-451, June 1974.

12. Goldberg, R. P., private communication, University of California, Los Angeles, California, March 1975.

13. Goldberg, R. P., Architectural Principles For Virtual Computer Systems, Ph.D. Thesis, Division Of Engineering And Applied Physics, Harvard University, Cambridge, MA., 1972.

14. Goldberg, R. P., "Architecture Of Virtual Machines," Proc. NCC, AFIPS Press., v. 42, p. 309-318, 1973.

15. Hawley, J. A. and Meyer, W. B., MUNIX, A Multiprocessing Version Of UNIX, M.S. Thesis, Computer Science Group, Naval Postgraduate School, Monterey, California, 1975.

16. Hoffman, L. J., "Computers And Privacy: A Survey," Computer Surveys, v. 1, no. 2, p. 85-97, June 1969.

17. Kral, T. C., A Process Controller For A Hierarchical Process Structured Operating System, Computer Science Group, Naval Postgraduate School, Monterey, California, 1975.

18. Laska, R. M., "Why All The Fuss About Virtual Computing?", Computer Decisions, p. 34, September 1972.

19. Madnick, S. E. and Donovan, J. J., Operating Systems, Mcgraw-Hill, New York, 1974.

20. Madnick, S. E., "Time-Sharing Systems: Virtual Machine Concept vs. Conventional Approach," Modern Data, v. 2, no. 3, p. 34-36, March 1969.

21. Marsh, M. T., Memory Management For Paged, Hierarchical Memory In A Multiprocessing Computer System, M.S. Thesis, Computer Science Group, Naval Postgraduate School, Monterey, California 1975.

22. Meyer, R. A. and Seawright, L. H., "A Virtual Machine Time-Sharing System," IBM Systems Journal, v. 9, no. 3, p. 119-218, 1970.

23. Popek, G. J. and Kline, C., "Verifiable Secure Operating System Software," Proc. NCC, AFIPS Press, v. 44, p. 145-152, 1974.

24. Popek, G. J. and Goldberg, R. P., "Formal Requirement For Virtualizable Third Generation Architectures," Communications Of The ACM, v. 17, no. 7, p. 412-421, July 1974.

25. Popek, G. J., "A Survey Of Protection Structures," Computer, v. 7, no. 6, June 1974.

26. Popek, G. J., private communication, University of California, Los Angeles, California, March 1975.

27. Popek, G. J., private correspondence, 11 April 1975.

28. Ritchie, D. M. and Thompson, K., "The UNIX Time-Sharing System," Communications Of The ACM, v.17, no. 7, p. 365-375, July 1974.

29. Thompson, K. and Ritchie, D. M., UNIX Programmer's Manual, 5th ed., Bell Telephone Laboratories, Incorporated, June 1974.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 72 Computer Science Group Naval Postgraduate School Monterey, California 93940	1
4.	LT. Belton E. Allen, Code 72An Computer Science Group Naval Postgraduate School Monterey, California 93940	1
5.	Professor G. L. Barksdale, Jr., Code 72Ba Computer Science Group Naval Postgraduate School Monterey, California 93940	1
6.	Major J. C. Winther, USMC 670 North 300 West American Fork, Utah 84003	1
7.	Captain D. M. Kruse, USMC Rural Route 1 Peterson, Iowa 51047	1